An Improved Firefly Algorithm for Mining High Utility Itemsets

Keerthi Mohan
Department of Computer Science and Engineering, RV Institute of Technology and Management, Bangalore, Karnataka, India | Visvesvaraya Technological University, Belagavi, Karnataka, India
keerthiimohan@gmail.com (corresponding author)

J. Anitha
Department of Computer Science and Engineering, RV Institute of Technology and Management, Bangalore, India
anithaj.rvitm@rvei.edu.in

Received: 28 April 2025 | Revised: 14 June 2025 and 19 June 2025 | Accepted: 21 June 2025

 DOI: https://doi.org/10.48084/etasr.11776

ABSTRACT

High-Utility Itemset Mining (HUIM) is a pivotal subfield of data mining that focuses on identifying itemsets with high utility rather than merely frequent patterns. Traditional HUIM algorithms are hindered by the exponential complexity of their search space, rendering them inefficient for large-scale databases. Evolutionary Computation (EC)-based approaches have emerged as promising alternatives, offering approximate, yet effective solutions. However, the existing EC-based HUIM methods still encounter considerable computational overhead when extracting true High-Utility Itemsets (HUIs). To address this constraint, this paper proposes a Firefly Algorithm (FA) and a modified FA for HUIM, incorporating a novel flexible inertia weight predicated on logarithmic decrement. This innovation leads to faster convergence and increased mining efficiency. The experimental evaluations on four publicly available benchmark datasets demonstrate that the proposed approach significantly outperforms state-of-the-art algorithms, including HUIM-Genetic Algorithm (GA), Hybrid Ant Colony Optimization (ACO)/GA, and HUIM-FA in terms of utility discovery, execution time, and scalability. The results indicate that the method is highly scalable and maintains robust performance across increasing transaction volumes and item dimensions. It achieves superior runtime, memory efficiency, and a high number of HUIs while sustaining a strong convergence rate, thus establishing its practicality for real-world, large-scale HUIM applications.

Keywords-high utility itemset mining; evolutionary computation; firefly algorithm; itemsets

I. INTRODUCTION

HUIM, unlike frequent itemset mining [1], emphasizes both the quantity and profit, making it a more meaningful approach for applications where the transactional utility is a key consideration. One of the early challenges in HUIM, namely, the effective discovery of useful patterns without being overwhelmed by the exponential number of item combinations, was addressed in [2], where a two-phase approach was proposed. However, this method still relied on candidate generation, limiting scalability. To address the computational inefficiencies, authors in [3] introduced the Utility Pattern Growth (UP-Growth) and UP-Growth+ algorithms, which improved the runtime by utilizing novel pruning strategies, such as utility upper bounds and transaction merging. These algorithms significantly reduced the computational complexity and demonstrated improved performance in large-scale databases.

Additionally, EC techniques have emerged as promising alternatives for tackling the extensive search space inherent in HUIM. GA-based methods, such as HUPEUMU-GARM and HUPEWUMU-GARM [4], have been developed to mine HUIs. While the latter relaxes the need for a minimum utility threshold, both approaches suffer from premature convergence, making them prone to local optima.

In response to the need for efficient HUIM algorithms, authors in [5] proposed Efficient High-Utility Itemset Mining (EFIM), which leveraged advanced pruning mechanisms to reduce the computational overhead and enhance the scalability. Similarly, authors in [6] introduced mHUIMiner, optimized for sparse datasets, using a novel data structure and refined search strategy to improve the mining speed.

Other metaheuristic algorithms have also been explored for HUIM. For instance, binary Particle Swarm Optimization (PSO)-based methods [7, 8] have utilized sigmoid functions and specialized data structures (e.g., OR/NOR trees) to guide

search trajectories ACO-based models [9, 12] have shown strong heuristic capabilities in navigating the complex solution space, while hybrid models combining ACO and GA have demonstrated additional flexibility. Moreover, authors in [10] employed a hybrid approach integrating Hill Climbing and Simulated Annealing to effectively balance exploration and exploitation, while authors in [13] developed HUIM-ICSO, which includes two variants, HUIM-CSA and HUIM-CSO, offering competitive results in terms of runtime, memory usage, and convergence rate. Additionally, authors in [14] proposed an incremental HUIM algorithm for dynamic databases using a utility tree, ensuring that HUIs are efficiently updated as new transactions arrive.

Despite the extensive application of EC-based techniques, the FA has not yet been applied to HUIM tasks. Originally designed for continuous optimization problems, FA is particularly well-suited for large-scale problems due to its ability to explore vast search spaces. An enhanced version of FA was proposed in [11], introducing a self-adaptive mechanism that dynamically adjusts parameters, such as attractiveness and movement step size, based on the search progress. This adaptive capability improves the convergence speed and accuracy, making FA a viable candidate for complex optimization problems.

This study adopts and extends the adaptive strategy from [11] to develop two novel algorithms for HUIM: a standard FA (HUIM-FA) and a modified FA (HUIM-Modified FA). These algorithms are specifically tailored to handle the discrete, combinatorial nature of HUIM tasks. The experimental results across multiple benchmark datasets demonstrate that both proposed methods outperform the existing state-of-the-art techniques in terms of runtime efficiency, memory utilization, and the number of HUIs discovered, offering a promising new direction for scalable and accurate utility-based pattern mining.

II. TERMINOLOGIES

Let $I = \{i_1, i_2, \dots\dots, i_j\}$ represent a finite set of j itemsets and $TD = \{T_1, T_2, \dots\dots, T_c\}$ represent a transaction database of c transactions, where each transaction T_c is a subset of I and has a unique identifier $c = (1 \le c \le j)$. The set $X \subseteq I$ is called an itemset, and if it contains k items, it is called a k-itemset. An itemset X is contained in a transaction T_c if $X \subseteq I$. Every item i_j in T_c has a positive number $q(i_j, T_c)$, called its internal utility, that represents the quantity (occurrence) of i_j in T_c. An additional positive figure known as the external utility $p(i_j)$ denotes the item's unit profit value. A profit table $ptable = \{p_1, p_2, \dots\dots, p_j\}$ denotes the profit value p_j of each item i_j.

1) Definition 1: Utility of an Item

The utility of an item i_j in a transaction T_c is defined as:

$$u(i_j, T_c) = p(i_j) \times q(i_j, T_c) \quad (1)$$

The utility of an itemset X in a transaction T_c, representing the total profit of X in a transaction T_c is given by:

$$u(X, T_c) = \sum_{i_j \in T_c \wedge X \subseteq T_c} u(i_j, T_c) \quad (2)$$

The overall utility of X in the database is computed as:

$$u(X) = \sum_{X \subseteq T_c \wedge T_c \in TD} u(X, T_c) \quad (3)$$

2) Definition 2: Transaction Utility

The transaction utility TU for a transaction T_c is defined as the utility of all items in that transaction:

$$TU(T_c) = u(T_c, T_c) \quad (4)$$

3) Definition 3: Minimum Utility Threshold

The minimum utility threshold δ is a user-defined percentage of the total utility of all transactions in the database. The corresponding minimum utility value is calculated as:

$$min\_util = \delta \times \sum_{T_c \in TD} TU(T_c) \quad (5)$$

4) Definition 4: High Utility Itemset (HUI)

An itemset X is considered as HUI, if $u(X) \ge \delta$.

5) Definition 5: Transaction Weighted Utility (TWU)

Transaction Weighted Utility (TWU) is used to reduce the search space [9]. The TWU of itemset X is the sum of the transaction utilities of all transactions that contain X:

$$TWU(X) = \sum_{X \subseteq T_c \wedge T_c \in TD} TU(T_c) \quad (6)$$

6) Definition 6: High Transaction Weighted-Utilization Itemset (HTWUI)

An itemset X s classified as a High Transaction Weighted-Utilization Itemset (HTWUI) if $TWU(X) \ge \delta$, otherwise, it is considered a Low Transaction Weighted-Utilization Itemset (LTWUI).

III. PROBLEM STATEMENT

The HUIM problem, as introduced in [12], is defined as follows: Given a transaction database TD, a profit table $ptable$, and a user-specified minimum utility threshold δ, the objective is to identify all itemsets whose utility is greater than or equal to δ. Table I represents the transaction database TD with associated transaction utility TU values, while Table II provides the profit table.

TABLE I. TRANSACTION DATABASE

TD	Transaction	TU
T00	(b, 2), (c, 10), (e, 2)	52
T01	(a, 4), (c, 2), (e, 1), (f, 5)	23
T02	(a, 3), (c,5), (d, 2), (e, 1)	20
T03	(b, 2), (d, 2), (f, 1)	25
T04	(a, 1), (c, 5), (d, 7)	52
T05	(a, 1), (d, 4), (f, 2)	24

TABLE II. PROFIT TABLE

Product	a	b	c	d	e	f
Profit	2	7	3	5	4	1

In this example, the utility of item b in transaction T00 is calculated as:

$$u(b, T00) = 7 \times 2 = 14$$

The utility of itemset $\{a, c\}$ across the database is given by:

$u(\{a,c\}) = u(\{a,c\},T01) + u(\{a,c\},T02) + u(\{a,c\},T04) = 14 + 21 + 17 = 52$

while the $TU(T00) = 7 \times 2 + 10 \times 3 + 2 \times 4 = 52$. If the $\delta = 80$, then itemset $\{a,c\}$ is not an HUI. However, the itemset is considered HTWUI since $TWU(\{a,c\}) = TU(T00) + TU(T02) + TU(T04) = 95 \geq \delta$.

IV. PROPOSED WORK

A. The Standard Firefly Algorithm

FA, first published in [15], is a metaheuristic inspired by the natural behavior of fireflies, particularly their flashing patterns used for communication and attraction. In the context of optimization, FA models each candidate solution as a firefly, whose brightness corresponds to its fitness value. Fireflies are attracted to brighter individuals, i.e., those representing better solutions. This mechanism drives the population toward regions of higher fitness within the solution space.

When applied to the HUIM problem, the brightness of a firefly is interpreted as the utility value of the corresponding itemset. Fireflies (solutions) are iteratively updated based on their relative attractiveness, which diminishes with distance, and a stochastic component that encourages exploration. The key FA parameters for HUIM include:

- Attractiveness β: Determines the degree to which a firefly is drawn to another.
- Light absorption coefficient γ: Controls the decay of attractiveness with increasing distance.
- Randomness factor α: Introduces stochasticity to avoid premature convergence.

The attractiveness of a firefly is given by:

$$\beta = \beta_0 \, e^{-\gamma \, r^2} \quad (7)$$

where β_0 is the initial attractiveness (typically set to 1), and r is the distance between two fireflies, which in HUIM, is measured using the Hamming distance between their corresponding binary itemset representations:

$$r_{ij} = \sum_{k=1}^{n} |x_{ik} - x_{jk}| \quad (8)$$

In accordance with the size of the dataset or the quantity of items in the transaction database, the γ is initialized. Moreover, to encourage the exploration of the search space, the randomization factor α is introduced and defined as:

$$\alpha = \alpha_0 \, \delta t \quad (9)$$

where α_0 is the initial randomness factor (typically set to 1), δ is a decay factor set to 0.95, and t is the current iteration. The movement of fireflies is governed by their attraction to brighter fireflies (higher fitness), which is described as:

$$X_i^{t+1} = X_i^t + \beta_0 \, e^{-\gamma \, r_{ij}^2} \left(X_j^t - X_i^t\right) + \alpha \, \varepsilon_t \quad (10)$$

where ε_t is a random perturbation.

1) Preprocessing and Initialization

Before applying FA, the transaction database undergoes a preprocessing phase involving encoding and pruning. A widely adopted technique for HUIM is the transformation of the transaction database into a binary bitmap representation [2]. In this scheme, the presence or absence of an item in a transaction is represented by bits "1" and "0", respectively. For an itemset X, its bitmap cover, denoted as $Bit(X)$, is computed using a bitwise-AND operation across the bitmap vectors of the constituent items. This operation yields a binary vector that identifies the transactions in which all items in X co-occur. The initial population for both the standard and enhanced versions of FA is randomly generated. All 1-HTWUIs are initially located by scanning the database, and 1-LTWUIs are removed since they cannot be a part of any HUI. The database is then converted into a bitmap. The initial candidate solutions are formed by randomly allocating a certain number of 1s (between 1 and the total number of 1-HTWUIs) to each member of the population.

B. The Modified Firefly Algorithm

To enhance the performance of the standard FA, the present work incorporated a self-adaptive inertia weight based on logarithmic decrement, as inspired by [14]. Inertia weights are commonly used in swarm intelligence algorithms, such as PSO, to regulate momentum, allowing a balance between the global exploration and local exploitation. This study embeds the inertia weight $w(t)$ into the firefly position update rule. This allows the influence of each firefly's previous position to gradually decay over time. The modified firefly movement equation is expressed as:

$$X_i^{t+1} = w(t) X_i^t + \beta_0 \, e^{-\gamma \, r_{ij}^2} \left(X_j^t - X_i^t\right) + \alpha \, \varepsilon_t \quad (11)$$

The $w(t)$ is designed to decay logarithmically with each iteration t, as:

$$w(t) = w_{max} - \frac{(w_{max} - w_{min})}{\log(t+1)+1} \quad (12)$$

where w_{max} and w_{min} are the initial and final inertia weights, respectively. Integrating this in (11) results in:

$$X_i^t = \left(w_{max} - \frac{(w_{max} - w_{min})}{\log(t+1)+1}\right) X_i^t + + \beta_0 \, e^{-\gamma \, r_{ij}^2} \left(X_j^t - X_i^t\right) + \alpha \, \epsilon_t \quad (13)$$

This formulation enables the adaptive control over the influence of past positions, promoting more effective convergence by dynamically adjusting the exploration–exploitation balance throughout the optimization process.

C. Algorithm Representation

The three pseudocode algorithms used in this study are presented below. Prior to population initialization, the transaction database must be preprocessed as outlined in Algorithm 1. The standard FA for mining HUIs is described in Algorithm 2, and the EFA with self-adaptive inertia weight is introduced in Algorithm 3.

```
Algorithm 1: Population initialization and
bitmap representation
```

Input: Transaction database TD, minimum utility value min_util, population size P, transaction T_c.

Step 1: Scan the database TD to calculate TWU for all 1-itemsets.
  1.1: Identify 1-HTWUL and eliminate 1-LTWUI, where
  1-HTWUI=items|TWU(item)>=min_util and
  1-LTWUI=items|TWU(item)<min_util
  1.2: Update TD by removing all 1-LTWUI
Step 2: Bitmap construction.
  2.1: Create a bitmap Bit(i) for item i
  2.2: for each transaction T_c in TD do
   If item i is present in T_c,
      Bit(i)=1
   Else
      Bit(i)=0
   End do
   End for
Step 3: Population initialization.
   3.1: Start a for loop from i = 1 to P
   3.2: Generate a random number n_i, an integer between 1 and |1-HTWUIs |
   3.3: Initialize all bits in item[j] to 0
   3.4: Randomly select a position in item[j] and set it to 1
   3.5: item[j] represents the candidate set
   End for
   End

Algorithm 2: Standard FA

Input: Transaction database TD, utility table U, number of fireflies or solutions N, initial attractiveness β_0, light absorption coefficient γ, randomness factor α, random perturbation factor ε, max_iter: Maximum number of iterations.

Step 1: Population initialization of itemsets by calling Algorithm 1.
Step 2: Determine the utility $U(X_i)$ of each firefly X_i by using the utility function over database transactions through (3).
Step 3: Calculate the Hamming distance between the respective itemsets which provides the distance r_{ij} between two fireflies, X_i and X_j using (8).
Step 4: Compare the utility as firefly X_i advances in the direction of a brighter firefly X_j due to the attractiveness β, which diminishes with increasing distance r_{ij}.
Step 5: After each movement, re-evaluate the utility for the new position of each firefly.
Step 6: Repeat steps 4 and 5 till max_iter is reached.
Step 7: End.

Algorithm 3: Modified FA

Input: Transaction database TD, utility table U, number of fireflies or solutions N, initial attractiveness β_0, light absorption coefficient γ, randomness factor α, random perturbation factor ε, max_iter: Maximum number of iterations, min_util: Threshold.

Step 1: Population Initialization using Algorithm 2, set the initial values for w_{max}, w_{min}, t (current iteration).
Step 2: Evaluate the fitness, for each firefly X_i calculate the utility function using (3).
Step 3: Introduce self-adaptive inertia weight into the firefly movement.
   3.1 Compute the distance between firefly r_{ij} through the Hamming distance.
   3.2 Update the position of firefly using (7).
Step 4: Evaluate the utility of the updated firefly (items), by comparing with the user-defined min_util value after each movement.
Step 5: Repeat steps 3 and 4 until the max_iter is reached.
Step 6: End.

V. PERFORMANCE EVALUATION

All experiments were conducted on a system equipped with an 8-core 3.6 GHz Central Processing Unit (CPU), 8 GB of Random Access Memory (RAM), and running Windows 10. The implementation was developed in Java.

A. Dataset

The performance evaluation was conducted on four widely used benchmark datasets: Mushroom, Chess, Connect, and Online Retail. These datasets, preprocessed for itemset mining, were sourced from the SPMF library and originally derived from the Frequent Itemset Mining Implementations (FIMI) repository and the UCI Machine Learning Repository [16–21]. Table III summarizes their key characteristics. The Mushroom dataset [17, 21] includes 8,124 instances representing mushroom species, each characterized by 22 categorical features (e.g., gill spacing, cap shape, odor). Each instance is

labeled as either edible or poisonous. For HUI mining, utility values (e.g., profit or importance) were assigned to specific features. The transformation from categorical to utility-based format posed challenges, requiring careful definition of the item and transaction utilities.

TABLE III. CHARACTERISTICS OF DATASETS

Dataset	Transaction length	No. of items	No. of transactions	Type
Mushroom	23	119	8,416	Dense
Chess	37	75	3,196	Dense
Connect	43	129	67,557	Dense
Online Retail	Variable	Variable	541,909	Sparse

The Online Retail dataset [18, 21] contains 541,909 records of a UK-based online retailer's transactions between December 2010 and December 2011. It includes both large and small transactions and presents issues, such as canceled orders and returns. Preprocessing was necessary to handle the negative quantities and outliers. Since utility values were not provided, they were computed manually using the formula: $Utility = quantity \times unit\ price$. The dataset is sparse and skewed, making it particularly suitable for testing the robustness of utility mining algorithms in practical scenarios. The Chess dataset [19, 21], structured as sequences of chess moves, is suitable for mining high-utility patterns, such as frequent or winning strategies. Each transaction represents a move sequence, and the utility values were associated with the importance or frequency of specific moves. Its regular structure and inherent repetitiveness make it ideal for evaluating HUI mining algorithms. The Connect dataset [20, 21], derived from the Connect-4 game, contains 67,557 instances and 43 categorical attributes. Each instance denotes a board configuration. For HUI mining, utility can be interpreted as the strategic value of certain move combinations. The dataset's size and structured nature support extensive benchmarking and pattern discovery.

B. Runtime Efficiency

Experiments were conducted to evaluate the runtime efficiency of the proposed algorithm (HUIM-Modified FA) against HUIM-FA, HUIM-GA, and HUIM-ACO/GA. The evaluation was performed by varying the minimum utility threshold, which directly impacts the computational load of HUIM. Table IV outlines the minimum utility thresholds selected for each dataset used in the experiments.

TABLE IV. MINIMUM UTILITY VALUES OF DATASETS

Dataset	Minimum utility threshold				
Chess	28.5	29	29.5	30	30.5
Mushroom	14	14.5	15	15.5	16
Connect	31.8	32	32.2	32.4	32.6
Online Retail	10	20	30	40	50

On the Chess dataset (Figure 1), HUIM-Modified FA consistently achieved the shortest execution time across all threshold values, outperforming the other three algorithms. A similar trend was observed on the Mushroom dataset (Figure 2), where HUIM-Modified FA again maintained the lowest execution times for all tested thresholds.

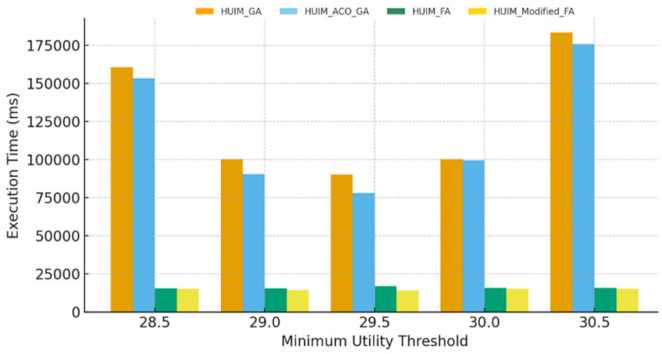

Fig. 1. Execution time: Chess dataset.

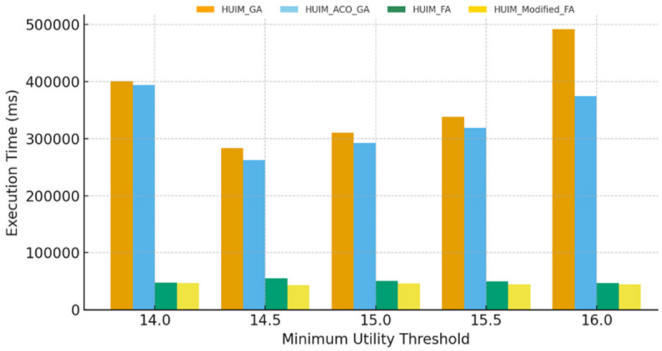

Fig. 2. Execution time: Mushroom dataset.

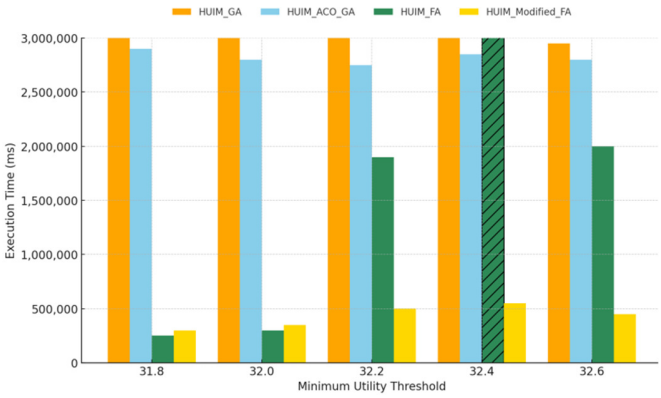

Fig. 3. Execution time: Connect dataset.

For the Connect dataset (Figure 3), HUIM-Modified FA demonstrated superior performance across all thresholds, highlighting its efficiency in optimizing the search process through balanced exploration and exploitation. Due to the large transaction volume in this dataset, HUIM-FA encountered extreme outliers at a minimum utility threshold of 32.4; these results were truncated and are displayed as hashed lines in the figure. In contrast, the GA-based approaches (HUIM-GA and HUIM-ACO/GA) showed stable execution times across all thresholds.

On the Online Retail dataset (Figure 4), which is sparse compared to the others, the GA-based algorithms achieved the shortest execution times across all thresholds.

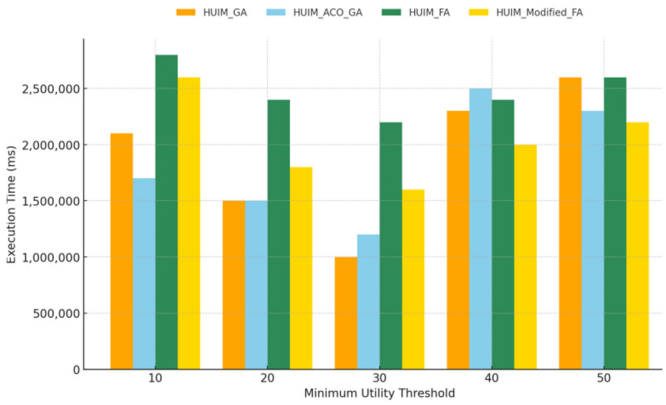

Fig. 4. Execution time: Online Retail dataset.

C. *Number of HUIs*

To assess the effectiveness of the proposed modified FA algorithm, this study compared the number of HUIs discovered across varying minimum utility thresholds.

As illustrated in Figure 5, at most thresholds, HUIM-ACO/GA consistently generates the greatest number of HUIs, peaking at approximately 280 HUIs at a threshold of 30.5%. On the Mushroom dataset (Figure 6), HUIM-Modified FA consistently mines the highest number of HUIs across all thresholds. A similar trend is observed in the Connect dataset (Figure 7), where HUIM-Modified FA again identifies the most HUIs at every threshold, confirming its strength in handling large-scale and complex data. HUIM-ACO/GA achieves moderate improvements, while HUIM-FA follows as the next best performer. HUIM-GA remains the least effective, highlighting its weaker exploratory capability in HUIM tasks.

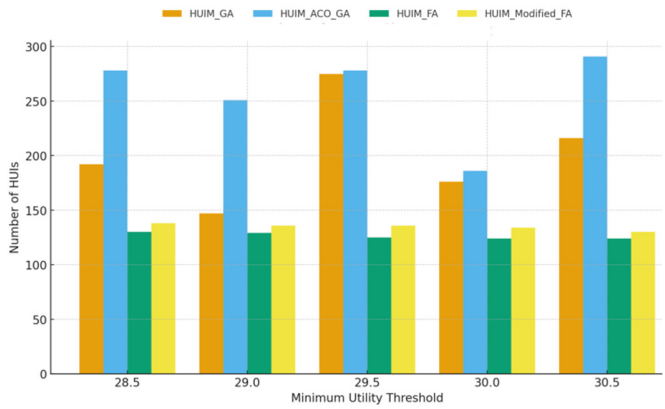

Fig. 5. Number of HUIs identified: Chess dataset.

For the Online Retail dataset (Figure 8), both HUIM-FA and HUIM-Modified FA produce the highest number of HUIs, with the modified version slightly outperforming the standard FA at higher thresholds. In contrast, HUIM-ACO/GA mines the fewest HUIs, while HUIM-GA maintains only a moderate level of performance.

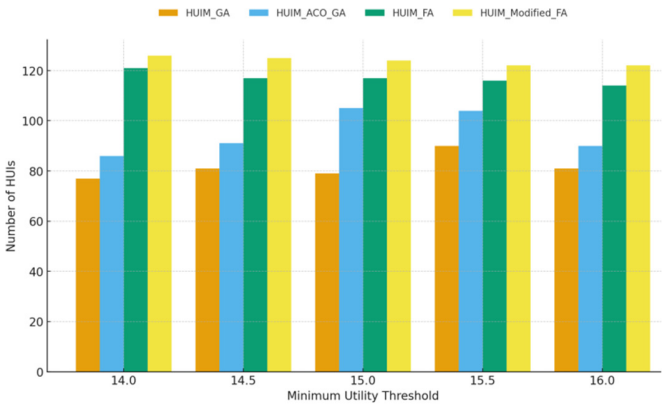

Fig. 6. Number of HUIs identified: Mushroom dataset.

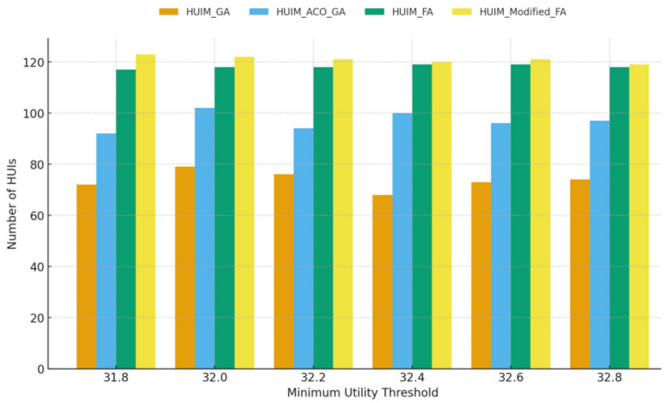

Fig. 7. Number of HUIs identified: Connect dataset.

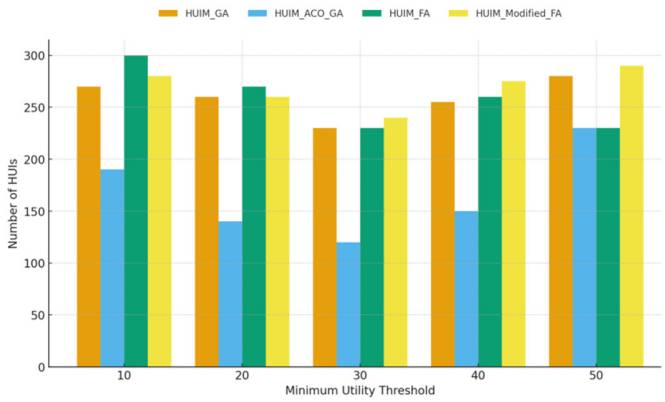

Fig. 8. Number of HUIs identified: Online Retail dataset.

These findings demonstrate the effectiveness of Firefly-based approaches in capturing additional HUIs across diverse datasets. In contrast, GA-based methods, although competitive in certain cases, remain comparatively limited. The dataset-specific variations further highlight the interplay between the algorithmic strategies and data characteristics. Specifically, HUIM-ACO/GA leverages genetic algorithm–driven diversification and pheromone-guided exploration to excel on dense, flat-utility datasets like Chess. Meanwhile, HUIM-Modified FA's self-adaptive parameters enhance the convergence and stability on large and sparse datasets, such as

Connect and Online Retail, enabling it to discover substantially more HUIs than GA-based approaches, which tend to stagnate. Overall, HUIM-Modified FA emerges as the preferred approach across diverse datasets, whereas HUIM-ACO/GA proves highly competitive on dense and structured datasets.

D. Memory

The efficiency of the modified FA algorithm was also evaluated in terms of the memory consumption across varying minimum utility thresholds and dataset sizes. The results are illustrated in Figures 9-12.

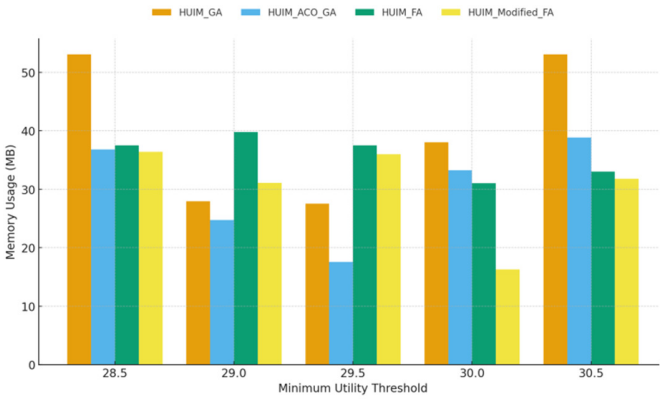

Fig. 9. Memory usage: Chess dataset.

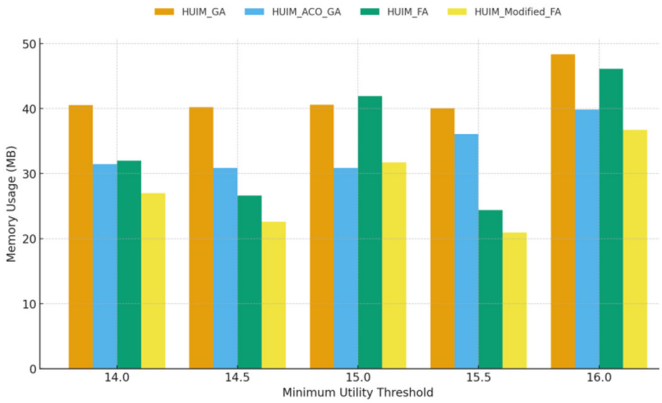

Fig. 10. Memory usage: Mushroom dataset.

There are clear differences among the four heuristic methods when comparing their memory usage across the Chess, Mushroom, Connect, and Online Retail datasets. HUIM-GA consistently exhibits the highest memory overhead, primarily due to its extensive population handling and chromosome storage requirements. HUIM-ACO/GA alleviates some of this burden, showing moderate memory consumption, although the additional pheromone maintenance still contributes to an increased demand. In contrast, the standard HUIM-FA is substantially more memory-efficient than the GA-based techniques, yet it remains less optimized than its modified version. The HUIM-Modified FA consistently achieves the lowest memory usage and maintains a stable performance, even on large datasets, such as Connect. On sparse datasets, like Online Retail, however, HUIM-ACO/GA demonstrates competitive efficiency, particularly at higher thresholds where it consumes less memory than other approaches.

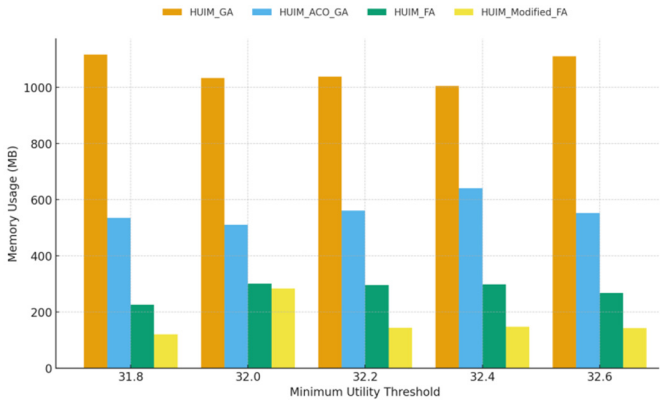

Fig. 11. Memory usage: Connect dataset.

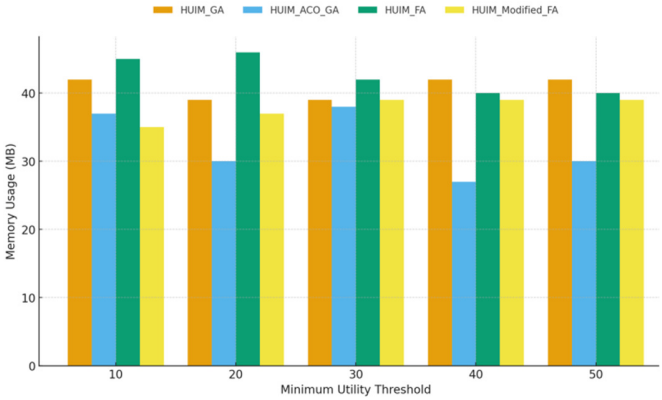

Fig. 12. Memory usage: Online retail dataset.

E. Convergence

To evaluate the performance of the proposed algorithm, a comparative analysis was conducted against four state-of-the-art HUIM algorithms: HUIM-GA, Hybrid ACO-GA, HUIM-FA, and HUI-Miner. The source codes for HUIM-GA and HUI-Miner were obtained from the open-source SPMF data mining library [21].

A runtime comparison was performed across varying minimum utility thresholds, which highlighted the distinct performance characteristics among the evaluated algorithms. The modified FA consistently achieved the lowest execution times, demonstrating superior computational efficiency across all scenarios. Although the standard FA also exhibited strong performance, the modified version provided a clear and consistent runtime advantage, particularly under stricter utility constraints. In contrast, HUI-Miner performed reasonably well but was notably slower than both FA-based approaches, especially as the minimum utility threshold increased. Among the evolutionary algorithms, HUIM-GA proved the least efficient, reflecting the limitations of conventional GAs when tackling the complexity of HUIM tasks without specialized enhancements. The Hybrid ACO-GA improved upon HUIM-

GA by leveraging the heuristic-driven search for more focused solution exploration, but it still lagged behind the FA-based approaches in runtime efficiency.

Overall, the findings indicate that HUIM-Modified FA is the most suitable algorithm for fast, real-time HUIM, whereas GA-based approaches may be better suited for exploratory analyses or scenarios where the execution time is less critical. These comparative results are depicted in Figures 13-16.

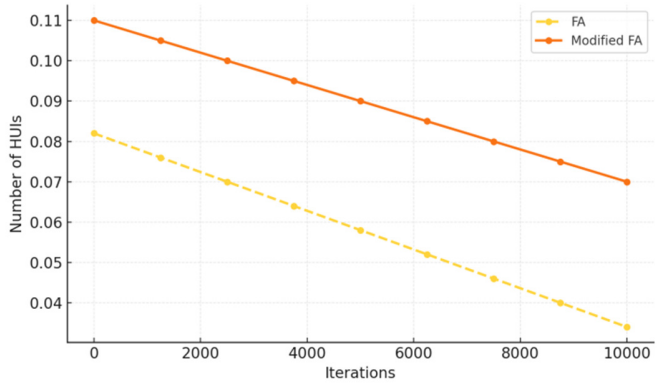

Fig. 13. Convergence rate: Chess dataset.

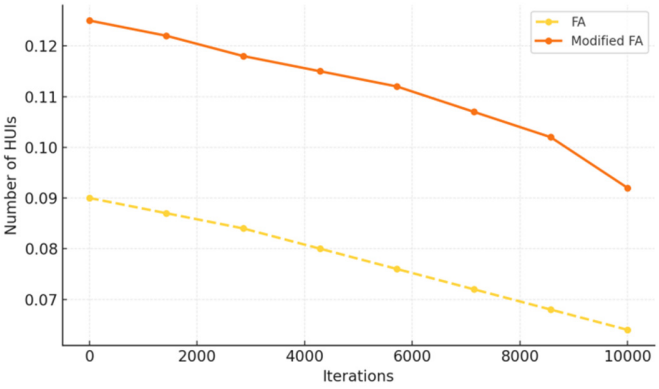

Fig. 14. Convergence rate: Mushroom dataset.

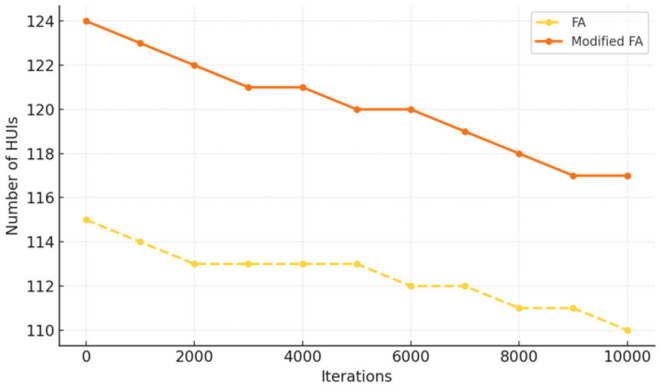

Fig. 15. Convergence rate: Connect dataset.

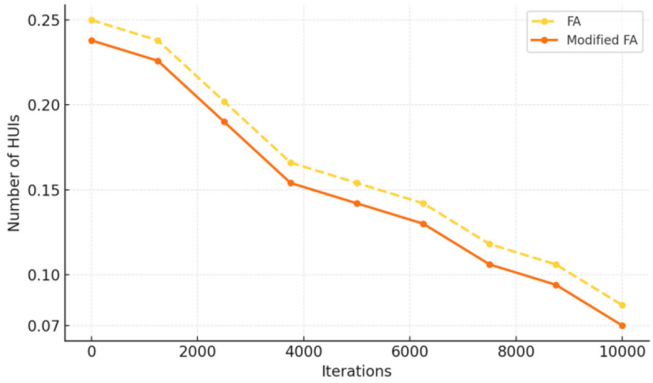

Fig. 16. Convergence rate: Online retail dataset.

VI. CONCLUSION

Finding frequent patterns in data is no longer sufficient in today's data-driven environment; what truly matters is identifying which patterns are useful and valuable. High-Utility Itemset Mining (HUIM) directly addresses this challenge by focusing on itemsets that contribute the greatest significance or profit. However, many traditional HUIM approaches struggle with speed, memory usage, and scalability when applied to large, complex, or dynamic datasets.

To overcome these challenges, this paper presented both the standard Firefly Algorithm (FA) for HUIM (HUIM-FA) and a modified Firefly Algorithm (HUIM-Modified FA) for HUIM. The proposed modified FA was evaluated against established HUIM algorithms, HUIM-Genetic Algorithm (GA), Hybrid Ant Colony Optimization (ACO)-GA, HUIM-FA, and HUI-Miner, through comparative runtime analysis. The experimental results demonstrated that HUIM-Modified FA consistently achieved superior runtime efficiency, particularly due to its self-adaptive inertia weight mechanism, which accelerates the convergence and minimizes the redundant computations. Although HUIM-GA and HUIM-ACO/GA perform competitively on sparse datasets at lower thresholds, they generally exhibit higher runtime values compared to the proposed modified FA.

Beyond the benchmark datasets, the proposed approach shows promise for real-world applications, including healthcare analytics, retail recommendation systems, and dynamic pricing. Furthermore, the algorithm can be adapted for streaming and real-time environments, where rapid and continuous pattern identification is essential. Future research could extend this work by integrating the HUIM-Modified FA with advanced optimization strategies or machine learning models, potentially leading to even more powerful HUIM solutions.

In conclusion, this study introduces a robust and adaptable method for HUIM, offering both theoretical and practical contributions, and opening new opportunities for the effective extraction of valuable patterns from complex data environments.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB '94: Proceedings of the 20th International*

Conference on Very Large Data Bases, San Francisco, CA, United States, Sep. 1994, pp. 487–499.

[2] Y. Liu, W. Liao, and A. Choudhary, "A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets," in *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2005, pp. 689–695, https://doi.org/10.1007/11430919_79.

[3] V. S. Tseng, B.-E. Shie, C.-W. Wu, and P. S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 8, pp. 1772–1786, Aug. 2013, https://doi.org/10.1109/tkde.2012.59.

[4] S. Kannimuthu and K. Premalatha, "Discovery of High Utility Itemsets Using Genetic Algorithm with Ranked Mutation," *Applied Artificial Intelligence*, vol. 28, no. 4, pp. 337–359, Apr. 2014, https://doi.org/10.1080/08839514.2014.891839.

[5] S. Zida, P. Fournier-Viger, J. C.-W. Lin, C.-W. Wu, and V. S. Tseng, "EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2015, pp. 530–546.

[6] A. Y. Peng, Y. S. Koh, and P. Riddle, "mHUIMiner: A Fast High Utility Itemset Mining Algorithm for Sparse Datasets," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2017, pp. 196–207.

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

[8] W. Song and J. Li, "Discovering High Utility Itemsets Using Set-Based Particle Swarm Optimization," in Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 38–53.

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

[10] K. Mohan and J. Anitha, "Mining High Utility Itemset with Hybrid Ant Colony Optimization Algorithm," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 12, 2024, https://doi.org/10.14569/ijacsa.2024.0151264.

[11] M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, and W. Song, "Mining High Utility Itemsets with Hill Climbing and Simulated Annealing," *ACM Transactions on Management Information Systems*, vol. 13, no. 1, pp. 1–22, Mar. 2022, https://doi.org/10.1145/3462636.

[12] M. Ledmi, A. Ledmi, M. E. H. Souidi, A. Hamdi-Cherif, T. M. Maarouk, and C. K.-M. Hamdi-Cherif, "High-utility itemsets mining integrating an improved crow search algorithm and particle search optimization," *Soft Computing*, vol. 28, no. 13–14, pp. 8471–8496, Jul. 2024, https://doi.org/10.1007/s00500-024-09758-0.

[13] S. R. Meruva and B. Venkateswarlu, "Dynamic Association Mining Techniques for the Faster Extraction of High Utility Itemsets from Incremental Databases," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19396–19400, Feb. 2025, https://doi.org/10.48084/etasr.9295.

[14] Y. Li, Y. Zhao, Y. Shang, and J. Liu, "An improved firefly algorithm with dynamic self-adaptive adjustment," *PLOS ONE*, vol. 16, no. 10, Oct. 2021, Art. no. e0255951, https://doi.org/10.1371/journal.pone.0255951.

[15] X.-S. Yang, *Nature-inspired metaheuristic algorithms*, 2nd ed. Frome: Luniver Press, 2010.

[16] R. J. Bayardo, "Efficiently mining long patterns from databases," *ACM SIGMOD Record*, vol. 27, no. 2, pp. 85–93, Jun. 1998, https://doi.org/10.1145/276305.276313.

[17] "Mushroom," UCI Machine Learning Repository, 1981. [Online]. Available: https://doi.org/10.24432/C5959T.

[18] D. Chen. "Online Retail II," UCI Machine Learning Repository, 2012. [Online]. Available: https://doi.org/10.24432/C5CG6D.

[19] M. Bain and A. Hoff. "Chess (King-Rook vs. King)," UCI Machine Learning Repository, 1994. [Online]. Available: https://doi.org/10.24432/C57W2S.

[20] J. Tromp. "Connect-4," UCI Machine Learning Repository, 1995. [Online]. Available: https://doi.org/10.24432/C59P43.

[21] P. Fournier-Viger *et al.*, "The SPMF Open-Source Data Mining Library Version 2," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2016, pp. 36–40.