This is the preprint of a paper that will appear in ACIIDS 2021.

Nawaz, M. S., Fournier-Viger, P., Song, W., Lin, J. C.-W., Noack, B. (2021). Investigating Crossover Operators in Genetic Algorithms for High-Utility Itemset Mining. Proc. of the 13th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2021), Springer LNAI, 12 pages

Investigating Crossover Operators in Genetic Algorithms for High-Utility Itemset Mining

M. Saqib Nawaz[1], Philippe Fournier-Viger[1],
Wei Song[2], Jerry Chun-Wei Lin[3], Bernd Noack[4]

[1]School of Humanities and Social Sciences,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2]School of Information Science and Technology, North China University of Technology, Beijing, China
[3]Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway
[4]Center for Turbulence Control,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
msaqibnawaz@hit.edu.cn, philfv8@yahoo.com, songwei@ncut.edu.cn, jerrylin@ieee.org, bernd.noack@hit.edu.cn

Abstract. Genetic Algorithms (GAs) are an excellent approach for mining high-utility itemsets (HUIs) as they can discover most of the HUIs in a fraction of the time spent by exact algorithms. A key feature of GAs is crossover operators, which allow individuals in a population to communicate and exchange information with each other. However, the usefulness of crossover operator in the overall progress of GAs for high-utility itemset mining (HUIM) has not been investigated. In this paper, the headless chicken test is used to analyze four GAs for HUIM. In that test, crossover operators in the original GAs for HUIM are first replaced with randomized crossover operators. Then, the performance of original GAs with normal crossover are compared with GAs with random crossover. This allows evaluating the overall usefulness of crossover operators in the progress that GAs make during the search and evolution process. Through this test, we found that one GA for HUIM performed poorly, which indicates the absence of well-defined building blocks and that crossover in that GA was indeed working as a macromutation.

Keywords: High-utility itemsets, Genetic Algorithms, Crossover.

1 Introduction

High-utility itemset mining (HUIM) [1,11,17,20] is a popular data mining problem, which aims at discovering all important patterns in a quantitative database, where pattern importance is measured using a numerical utility function. One of the main applications of HUIM is to enumerate all the sets of items (itemsets) purchased together that yield a high profit in customer transactions. An itemset is called a high-utility itemset (HUI) if its utility (profit) value is no less than a user-specified minimum utility threshold. Several exact HUIM algorithms have

been designed to efficiently find all HUIs. However, these algorithms can still have very long runtimes because the search space size is exponential with the number of distinct items [1]. Long runtimes are inconvenient for users who often have to wait hours to obtain results even on small databases.

To address this issue, an emerging research direction is to design Nature-inspired Algorithm (NAs) for HUIM as they can solve hard optimization and computational problems. NAs have been used for problems ranging from bioinformatics and scheduling applications to artificial intelligence and control engineering. Some of the most popular NAs are Genetic Algorithms (GAs) [4], Simulated Annealing (SA) [8] and Particle Swarm Optimization (PSO) [7]. NAs were proposed to find HUIs in large databases based on GAs [6, 9, 15, 21], PSO [9, 10, 15, 16], Artificial Bee Colony (ABC) [14], the Bat algorithm [15] and Ant Colony System (ACS) [18]. In this paper, we are interested by GAs as they have excellent performance, are easy to implement and can discover most of the HUIs in a fraction of the time spent by exact algorithms.

A key feature that differentiates GAs from other NAs is crossover operators. The crossover process is simple: two chromosomes (solutions) are selected as parents and parts of them are combined to generate a new solution. The main *idea* of crossover is that such combination may yield better child solutions. This intuition was formalized by Holland [4] with the concept of building blocks used in schema theory. The *mechanics* of crossover provides a way to implement this *idea*. Thus, all types of crossover share the same *idea* but the *mechanics* to implement the *idea* can vary considerably. For example, single-point crossover (SPC) uses a single crossing point while two-point crossover (TPC) uses two.

Despite that GAs provide excellent performance for HUIM, the influence of crossover for that problem has not been investigated. Assessing the usefulness of crossover operators in GAs is an important research topic. Jones [5] argued that crossover *mechanics* alone can be used effectively for search and evolution even in the absence of the crossover *idea*. For this, a testing method (called headless chicken test) was proposed to examine the usefulness of crossover for a particular problem instance. In that test, a normal GA is compared with the same GA that uses a random version of crossover. Using this test, one can distinguish the gains that the GA makes through the idea of crossover from those made simply through the mechanics. If GA is not making any additional progress due to the idea of crossover, one might do as well simply by using macromutations. A poor performance of the original GA with normal crossover compared to the GA with random crossover indicates the absence of well-defined building blocks.

In this paper, we perform the headless chicken test to assess the usefulness of crossover in GAs for HUIM to better understand their performance. We applied the test on four GAs for HUIM [6, 9, 15, 21]. In the test, original GAs for HUIM (called normal GAs) were compared with the same GAs with randomized crossover operators (called randomized GAs). We found that three normal GAs for HUIM [6, 9, 21] that use normal crossover such as SPC and uniform crossover performed almost the same as randomized GAs. The GA for HUIM [15] that defined a new crossover performed worse than the randomized GA.

The remainder of this paper is organized as follows. Section 2 briefly discusses HUIM and GAs respectively. Section 3 provides the details for the headless chicken test performed on four GAs for HUIM. Section 4 presents the experiments performed in the test and discusses the obtained results. Finally, the paper is concluded with some remarks in Section 5.

2 Preliminaries

This section introduces preliminaries about high-utility itemset mining and genetic algorithms.

High-utility itemset mining. Let $I = \{i_1, i_2, ..., i_m\}$ be a finite set of m distinct items and $TD = \{T_1, T_2, T_3, ..., T_n\}$ be a transaction database. Each transaction T_c in TD is a subset of I and has a unique integer identifier c ($1 \leq c \leq n$) called its TID. A set $X \subseteq I$ is called an *itemset* and an itemset that contains k items is called a k-itemset. Every item i_j in a transaction T_c has a positive number $q(i_j, T_c)$, called its *internal utility*. This value represents the purchase quantity (occurrence) of i_j in T_c. The *external utility* $p(i_j)$ is the unit profit value of the item i_j. A profit table $ptable = \{p_1, p_2, ..., p_m\}$ indicates the profit value p_j of each item i_j in I.

The overall utility of an item i_j in a transaction T_c is defined as $u(i_j, T_c) = p(i_j) \times q(i_j, T_c)$. The *utility of an itemset X in a transaction T_c* is denoted as $u(X, T_c)$ and defined as $u(X, T_c) = \sum_{i_j \subseteq X \wedge X \subseteq T_c} u(i_j, T_c)$. The *overall utility of an itemset X* in a database TD is defined as $u(X) = \sum_{X \subseteq T_c \wedge T_c \in TD} u(X, T_c)$ and represents the profit generated by X.

The *transaction utility* (TU) of a transaction T_c is defined as $TU(T_c) = u(T_c, T_c)$. The *minimum utility threshold* δ, specified by the user, is defined as a percentage of the sum of all TU values for the input database, whereas the *minimum utility value* is defined as $min\_util = \delta \times \sum_{T_c \in TD} TU(T_c)$. An itemset X is called an HUI if $u(X) \geq min\_util$.

The problem of HUIM is defined as follows [19]. Given a transaction database (TD), its profit table ($ptable$) and the minimum utility threshold, the goal is to enumerate all itemsets that have utilities equal to or greater than min_util.

To reduce the search space in HUIM, an upper bound on the utility of an itemset and its supersets called the *transaction-weighted utilization* (TWU) is often used [11]. The TWU of an itemset X is the sum of the transaction utilities of all the transactions containing X, which is defined as $TWU(X) = \sum_{X \subseteq T_c \wedge T_c \in TD} TU(T_c)$. An itemset X is called a *high transaction weighted-utilization itemset* (HTWUI) if $TWU(X) \geq min\_util$; otherwise, X is a low transaction weighted-utilization itemset (LTWUI). An HTWUI/LTWUI with k items is called a k-HTWUI/k-LTWUI.

Genetic Algorithms. GAs [4] are based on Darwin's theory (survival of the fittest) and biological evolution principles. GAs have the ability to explore a huge search space (population) to find nearly optimal solutions to difficult problems that one may not otherwise find in a lifetime. The foremost steps of a GA include: (1) population generation, (2) selection of candidate solutions from a

population, (3) crossover and (4) mutation. Candidate solutions in a population are known as chromosomes or individuals, which are typically finite sequences or strings ($x = x_1, x_2 ..., x_n$). Each x_i (genes) refers to a particular characteristics of the chromosome. For a specific problem, GA starts by randomly generating a set of chromosomes to form a population and evaluates these chromosomes using a fitness function f. The function takes as parameter a chromosome and returns a score indicating how good the solution is. The general framework of GAs to mine HUIs is shown in Figure 1. For HUIM, the utility of an itemset is used as the fitness function and the stopping criterion is the user-specified maximum number of generations.

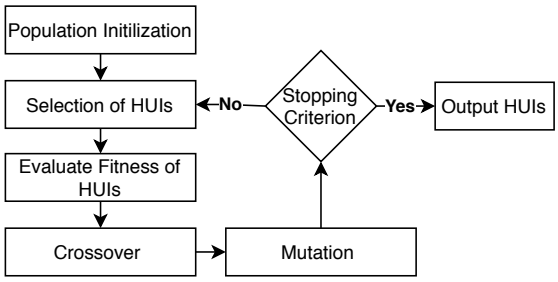

Fig. 1: General framework of GAs for HUIM

The crossover operator is used to guide the search toward the best solutions. If an appropriate crossing point is chosen, then the combination of sub-chromosomes from parent chromosomes may produce better child chromosomes. The mutation operator applies some random changes to one or more genes. This may transform a chromosome into a better chromosome.

3 The Headless Chicken Test for HUIM using GAs

In this study, we assess the usefulness of crossover operators in GAs for the HUIM problem by applying the headless chicken test. In that test, a GA with normal crossover is compared with the identical GA with random crossover. The random crossover operator is illustrated in Figure 2. While a normal crossover combines two parents to generate two new individuals (using either SPC or TPC), a random crossover generates two random individuals and uses them to do crossover with the parents. In the random crossover, there is no direct communication between parents. As individuals involved in random crossover are randomly selected, the operation is purely mechanical and does not carry the spirit of crossover. Despite the identical mechanical rearrangement, it is argued that random crossover is not a crossover [5]. A reason is that this operation does not requires two parents. For example, for TPC, one can simply select the crossing points and set the loci between the points to randomly chosen alleles. Hence, random crossover is a macromutation.

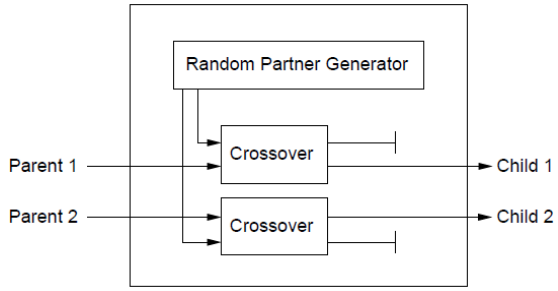

Fig. 2: Headless chicken test with random crossover [5]

This study applies the headless chicken test to four GAs implementations for HUIM: HUIM-GA [6], HUIM-GA-tree [9], HUIF-GA [15] and HUIM-IGA [21]. The original version of each GA is compared with a random crossover version. In all the four algorithms, each chromosome (solution) represents a set of items that form a potential HUI. Each chromosome is composed of binary values (0 or 1) that tell whether an item is absent or present in the chromosome. A value of 1 at the i-th position of a chromosome means that the corresponding item is present in the potential HUI, while a value of 0 indicates that it is absent. The total number of 1-HTWUIs in the database represent the size of the chromosome.

In HUIM-GA [6] two GAs (called HUPE$_{UMU}$-GARM and HUPE$_{WUMU}$-GARM) were proposed for HUIM. Applying HUPE$_{UMU}$-GARM requires setting a minimum utility threshold while HUPE$_{WUMU}$-GARM does not require a minimum utility threshold. Both GAs used the common operators (selection, single-point crossover, and mutation) iteratively to find HUIs.

Let there be two parent chromosomes $x = x_1, x_2, ..., x_n$ and $y = y_1, y_2, ..., y_n$ of length n [12]. Let position i ($1 \leq i \leq n$) be a randomly selected crossing point in both parent chromosomes. The two new child chromosomes generated by the single-point crossover operator with i are:

$$x' = x_1, ..., x_i, y_{i+1}, ..., y_n$$
$$y' = y_1, ..., y_i, x_{i+1}, ..., x_n$$

Differently from single-point crossover, two-point crossover selects two crossing points i, j such that $1 \leq i \leq j \leq n$. The result is two new child chromosomes:

$$x' = x_1, ..., x_i, y_{i+1}, ..., y_j, x_{j+1}, ...x_n$$
$$y' = y_1, ..., y_j, x_{i+1}, ..., x_k, y_{j+1}, ...y_n$$

Examples of SPC and TPC, and their randomized versions are presented in Figure 3(a) and Figure 3(b), respectively. There, P_1, P_2 represent two parents, and C_1, C_2 represent the generated childs and R_1, R_2 are the random solutions.

HUIM-GA [6] cannot easily find the 1-HTWUIs initially to use them as chromosomes and thus perform a very large search for selecting appropriate

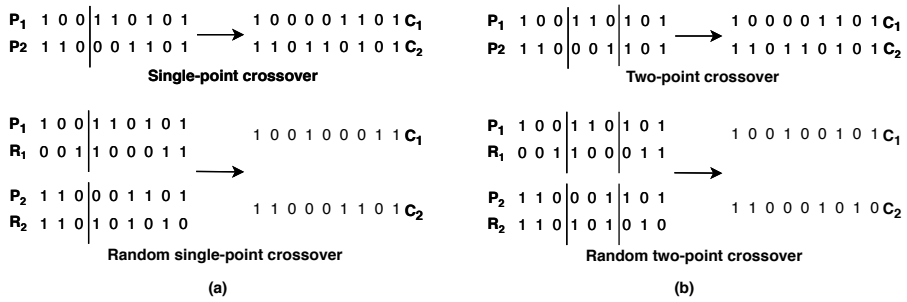

Fig. 3: SPC and TPC with randomized versions

chromosomes for mining valid HUIs. Additionally, setting the appropriate values for some specific parameters is a nontrivial task. The performance of HUIM-GA [6] was improved in HUIM-GA-tree [9] by using the OR/NOR-tree structure for pruning. SPC was also used in HUIM-GA-tree.

HUIF-GA [15] makes use of efficient strategies for database representation and a pruning process to accelerate HUI discovery. This GA does not use any normal crossover version such as SPC or TPC. For crossover, HUIF-GA uses a technique called *BittDiff* to first find the locations in two parents (bit vectors) where their values do no match. *BittDiff* is defined as follows:

Definition 1. *Let P_1 and P_2 be the two bit vectors with n bits. The bit difference set is defined as [15]:*

$$BitDiff(P_1, P_2) = \{i | 1 \leq i \leq n, b_i(P_1) \oplus b_i(P_2) = 1\} \tag{1}$$

where $b_n(P_1)$ is the n-th bit of P_1 and $\oplus$ denotes the exclusive disjunction.

The total number of points that will be used for crossover in the two selected chromosomes is:

$$cnum = \lfloor |BittDiff(P_1, P_2)| \times r \rfloor \tag{2}$$

where r is a random number in the range (0, 1), $|BittDiff(P_1, P_2)|$ is the number of elements in $BitDiff(P_1, P_2)$, and $\lfloor |BittDiff(P_1, P_2)| \times r \rfloor$ denotes the largest integer that is less than or equal to $|BitDiff(P_1, P_2)| \times r$. We call this crossover *BittDiff* crossover (BDC). It is explained with a simple example. Suppose that $P_1 = 10010$ and $P_2 = 11000$. As $10010 \oplus 11000 = 01010$, $BitDiff(P_1, P_2) = \{2, 4\}$, that is, P_1 and P_2 differ in their second and fourth positions. Suppose that the random number r is 0.5. Then, $cnum = \lfloor (2 \times 0.5) \rfloor = \lfloor 1 \rfloor = 1$. For P_1, one position (either the second or fourth) will be selected for crossover. Suppose the second position is selected to be changed. Then, the new P_1 is 11010 and new P_2 is 10000.

In the randomized version of BDC, two random sequences (R_1 and R_2) are provided as input to *BitDiff*, in place of the two parents. So the place for crossover in parents is determined by using *BitDiff* on random sequences. Suppose $P_1 = 10101$ and $P_2 = 10001$, $R_1 = 10001$ and $R_2 = 11010$. Then, $BitDiff(R_1, R_2) =$

$\{2, 4, 5\}$. Suppose the random number r is 0.6, $cnum = \lfloor (3 \times 0.6) \rfloor = \lfloor 1.8 \rfloor = 1$. Then, the new $P_1 = 11101$ and $P_2 = 10011$.

Another GA for HUIM, named HUIM-IGA [21] introduced many novel strategies to efficiently mine HUIs. It employs a uniform crossover (UC) operator along with single-point mutation. In UC, each element (gene) of the first parent chromosome is assigned to a child chromosome with a probability value p, and the rest of the genes are selected from the second parent. For instance, if $p = 0.5$, the child has approximately half of the genes from the first parent and the other half from the second parent [13]. The randomized version of UC is depicted in Figure 4. For UC, generated child chromosomes can be different for each run as it depends on the selection probability.

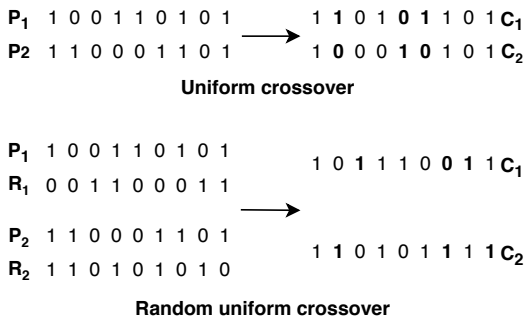

Fig. 4: Uniform crossover and its randomized version

Note that in HUPE$_{UMU}$-GARM and HUIM-GA-tree, we replaced SPC by random SPC and random TPC. The reason is to compare these two GAs with two randomized versions to check whether the change in the crossover operator has an effect on the overall performance of these two GAs.

4 Experiments and Results

This section presents the experimental evaluation of the headless chicken test on the four GAs. The experiments were performed on a computer with an 8-Core 3.6 GHz CPU and 64 GB memory running 64-bit Windows 10. The programs for randomized GAs were developed in Java. Five real standard benchmark datasets were used to evaluate the performance of the algorithms. The Foodmart dataset has real utility values while the remaining four datasets have synthetic utility values. The characteristics of the datasets are presented in Table 1.

All the datasets were downloaded from the SPMF data mining library [2]. The Foodmart dataset contains customer transactions from a retail store. The Chess dataset originates from game steps. The Mushroom dataset describes various species of mushrooms and their characteristics, such as shape, odor, and habitat. The Accident dataset is composed of (anonymized) traffic accident data. Similar

Table 1: Characteristics of the datasets

Dataset	Avg. Trans. Len.	#Items	#Trans	Type
Foodmart	4.42	1,559	4,141	Sparse
Chess	37	75	3,196	Dense
Mushroom	23	119	8,124	Dense
Accidents_10%	34	468	34,018	Dense
Connect	43	129	67,557	Dense

to previous studies [9,14,15,21], only 10% of this dataset was used in experiments. The Connect dataset is also derived from game steps. For all experiments, the termination criterion was set to 10,000 iterations and the initial population size was set to 30.

In the experiments, the normal GA (that uses SPC) for HUIM [6] is called HUPE$_{UMU}$-GARM and the HUPE$_{UMU}$-GARM with random SPC and random TPC are named HUPE$_{UMU}$-GARM+ and HUPE$_{UMU}$-GARM++ respectively. The normal GA (that uses SPC) for HUIM in [9] is named HUIM-GAT and HUIM-GAT with random SPC and random TPC are named HUIM-GAT+ and HUIM-GAT++, respectively. Similarly, the original GA that uses BDC for HUIM in [15] is named HUIF-GA and the HUIF-GA with random BDC is called HUIF-GA+. The original GA for HUIM in [21] is named HUIM-IGA and the HUIM-IGA with random uniform crossover is called HUIM-IGA+.

4.1 Runtime

Experiments were first carried out to evaluate the efficiency of the algorithms in terms of runtime. The runtime was measured while varying the minimum utility value for each dataset. Figure 5 shows the execution time of algorithms for the five datasets.

It is observed that the randomized HUIF-GA+ algorithm was slower than the normal HUIF-GA for all datasets. Moreover, it was slower than all other algorithms except for the Foodmart database, where HUIF-GA+ was faster than HUIM-GAT and its randomized versions (HUIM-GAT+ and HUIM-GAT++) at the start. However, HUIF-GA+ tends to become slower than all other algorithms as the minimum utility value is increased. On the other hand, all other algorithms and their randomized versions have almost the same execution time with negligible difference. Almost the same execution times for HUPE$_{UMU}$-GARM and HUIM-GAT and their randomized versions suggest that the use of SPC or TPC has no noticeable effect on the runtime of these algorithms. On the basis of runtime, HUIM-IGA and its randomized version (HUIM-IGA+) performed better than other algorithms on five datasets.

On the Foodmart dataset, HUPE$_{UMU}$-GARM and its randomized versions did not terminate after more than five hours. That is why their results are not shown in the chart. For the same reason, results of HUIM-GAT and its

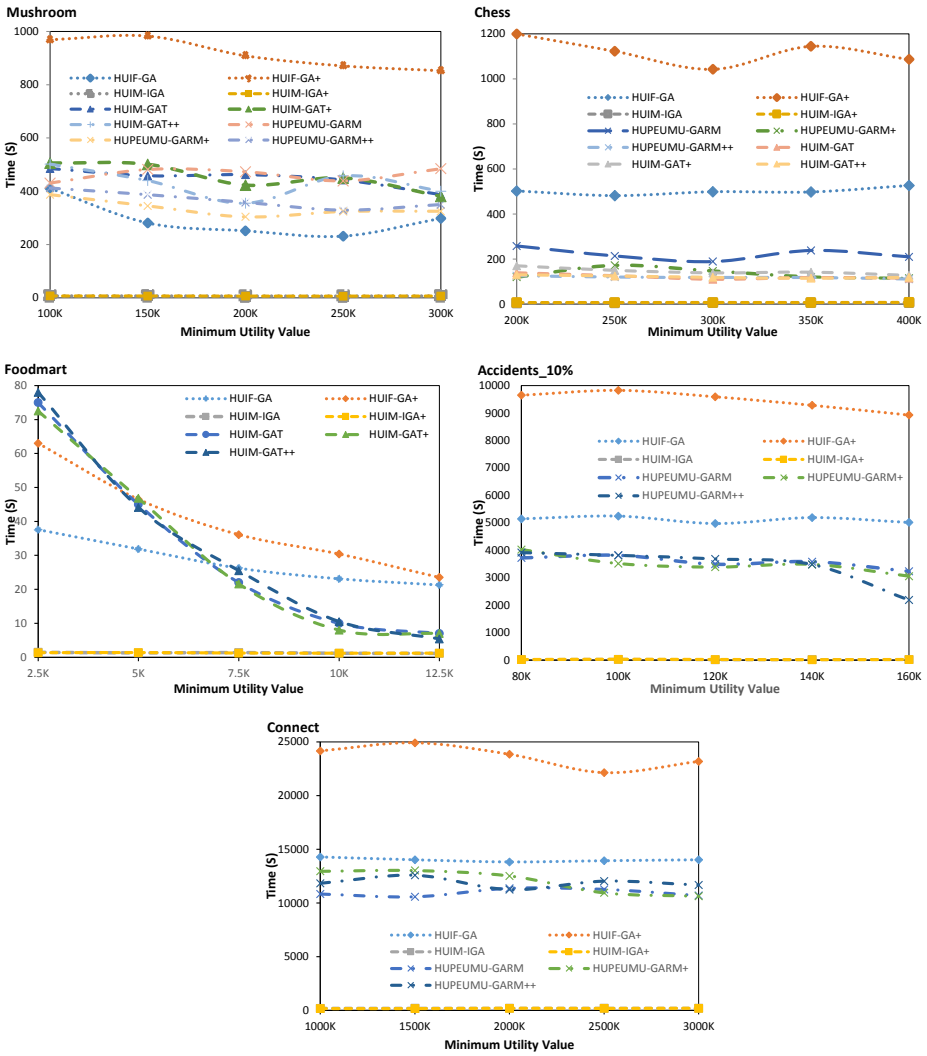

Fig. 5: Execution times of compared algorithms on five datasets

randomized versions are not shown for the Accidents and Connect datasets. They were unable to terminate in less than ten hours.

4.2 Discovered HUIs

Next, the numbers of HUIs discovered by the algorithms for the five datasets and parameter values are compared. The results are shown in Table 2.

HUIF-GA+ outperformed HUIF-GA on all datasets. This seems to be the reason why HUIF-GA+ was slower than the other algorithms. The performance

Table 2: Discovered HUIs

D	MUV	HUIF-GA/GA+	HUIM-IGA/IGA+	GARM/GARM+/GARM++	HUIM-GAT/GAT+/GAT++
M	100K	15349/24846	5948/6221	66/75/96	68/78/94
	150K	10714/18945	4490/5450	57/53/85	61/62/75
	200K	8796/ 12285	3630/4568	45/31/65	47/43/57
	250K	6637/11731	3082/3791	33/28/51	34/21/24
	300K	5031/9498	2821/3534	22/16/47	23/31/15
Ch	200K	18269/33030	6773/6872	205/198/250	172/185/173
	250K	17501/29078	5912/6142	184/157/205	137/142/138
	300K	14151/25596	5120/5619	153/133/163	105/116/110
	350K	13437/21148	4814/4954	128/94/130	85/95/94
	400K	9828/17654	4211/4313	96/71/ 89	59/70/85
F	2.5K	2076/2436	1493/1515	x/x/x	72/77/75
	5K	921/1153	1085/1084	x/ x/x	41/46/40
	75.5K	508/748	707/705	x/x/x	18/19/16
	10K	312/465	424/423	x/x/x	5/4/0
	12.5K	196/236	207/209	x/x/x	0/0/0
A	80K	84856/98256	4836/4829	433/445/437	x/x/x
	100K	80214/90365	5021/4942	384/385/378	x/x/x
	120K	74785/84561	4694/4620	314/307/311	x/x/x
	140K	68646/76431	4482/4392	257/272/267	x/x/x
	160K	61915/69752	4577/4588	207/216/214	x/x/x
Co	1000K	58925/66154	5910/5879	137/142/147	x/x/x
	1500K	52589/59745	5864/5814	102/107/105	x/x/x
	2000K	48254/53856	5801/5694	88/92/98	x/x/x
	2500K	45812/52380	5635/5604	57/55/61	x/x/x
	3000K	40987/46982	5546/5484	32/34/37	x/x/x

D = Dataset, MUV = Minimum utility value, GARM = HUPE$_{UMU}$-GARM, Mushroom, Ch = Chess, F = Foodmart, A = Accidents, Co = Connect, x = run out of time

of other normal GAs and their randomized versions were almost similar with negligible difference. The same performance of HUPE$_{UMU}$-GARM, HUIM-GAT, HUIM-IGA and their randomized versions indicate the presence of building blocks for the crossover operators (SPC and UC). On the other hand, the BDC in HUIG-GA was indeed working as a macromutation and failed to incorporate the basic idea of crossover. The reason for this is the non-availability of building blocks that makes HUIF-GA to perform more poorly than HUIF-GA+. BDC failed to implement the crossover idea of exchanging the building blocks between individuals as it was changing the values of bits in individuals (HUIs) at specific locations.

Note that for the Foodmart dataset, the results for HUPE$_{UMU}$-GARM and its randomized versions (HUPE$_{UMU}$-GARM+ and HUPE$_{UMU}$-GARM++) were not included because they were unable to terminate after five hours of execution. For the same reason, no results is shown for HUIM-GAT on Accidents and Connect. HUIM-GAT and its randomized version were unable to terminate after ten hours of execution. Lastly, HUPE$_{UMU}$-GARM and HUIM-GAT and their randomized versions discovered almost the same number of HUIs. This suggests that the use of SPC or TPC has the same effect on the performance of these two algorithms in terms of discovered HUIs.

4.3 Convergence

As HUIF-GA+ outperformed HUIF-GA, we evaluate their convergence speed for all datasets. Obtained results are shown in Figure 6.

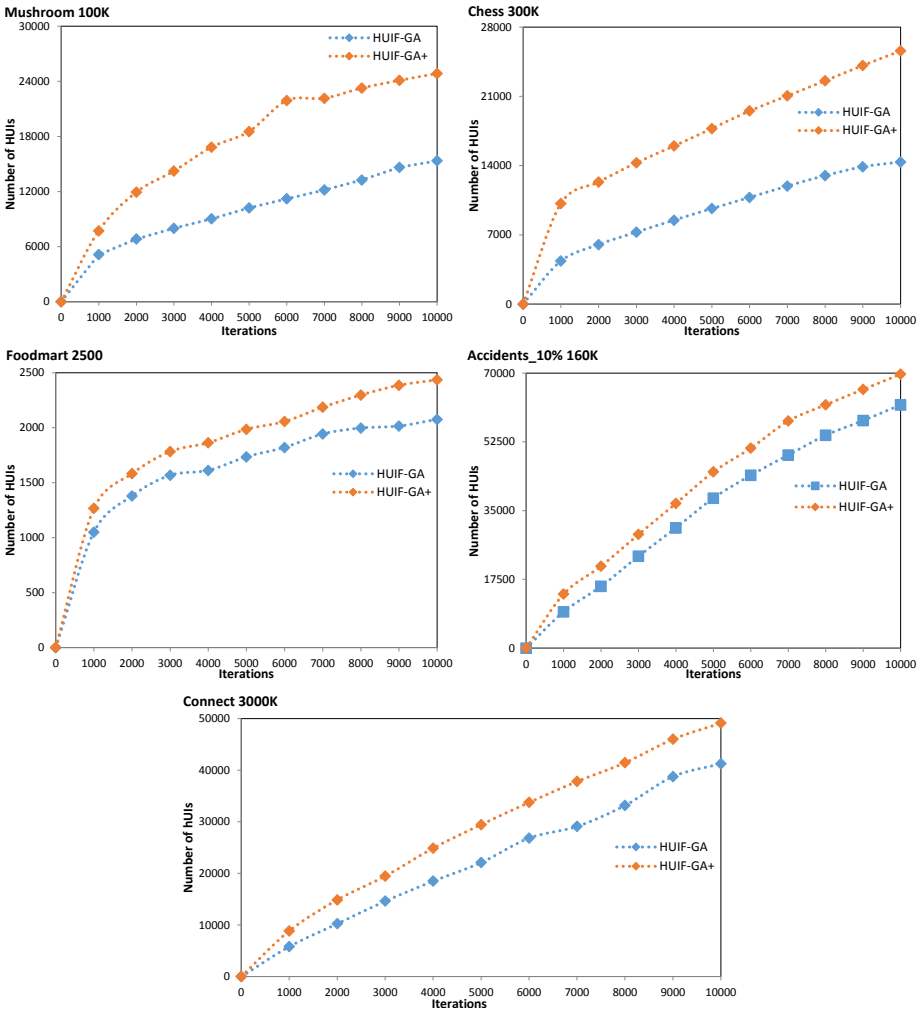

Fig. 6: Convergence performance of HUIF-GA and HUIF-GA+

HUIF-GA+ converged faster than HUIF-GA from the start on all datasets. The convergence speed of HUIF-GA and HUIF-GA+ for the Accidents and Connect datasets that contain a large number of transactions were linear. Whereas their convergence speed on other datasets (Mushroom, Chess and Foodmart) that have much less transactions (compared to Accidents and connect) were lin-

ear at the start (in the first 1000 iterations). However, as the number of iterations was increased, the convergence speed of both algorithms tend to decrease.

5 Conclusion

This paper investigated the performance of four GAs for HUIM by performing the headless chicken test. This test investigates the usefulness and worth of crossover operators in GA. Obtained results showed that three GAs for HUIM that employed normal crossover (such as single-point crossover and uniform crossover) were helping GAs to make progress. However one GA that was not employing any normal crossover was actually working as a macromutation that indicated the absence of well-defined building blocks. Thus, the efficacy of new crossover operators, particularly the specialized one for HUIM, can be investigated with the headless chicken test.

In the future, we intend to implement the PSO algorithms with headless chicken macromutation [3] for HUIM and compare the results with already implemented PSO algorithms to mine HUIs [9, 10, 15].

References

1. P. Fournier-Viger, J. Chun-Wei Lin, T. Truong-Chi, and R. Nkambou. *A survey of high utility itemset mining*, pages 1–45. Springer, 2019.
2. P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF open-source data mining library version 2. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 36–40. Springer, 2016.
3. J. Grobler and A. P. Engelbrecht. Headless chicken particle swarm optimization algorithms. In *Intl. Conference on Swarm Intelligence*, pages 350–357. Springer, 2016.
4. J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, USA, 1975.
5. T. Jones. Crossover, macromutation, and population-based search. In *6th Intl. Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
6. S. Kannimuthu and K. Premalatha. Discovery of high utility itemsets using genetic algorithm with ranked mutation. *Applied Artificial Intelligence*, 28(4):337–359, 2014.
7. J. Kennedy and R. Eberhart. Particle swarm optimization. In *Intl. Conference on Neural Networks*, pages 1942–1948. IEEE, 1995.
8. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
9. J. C. Lin, L. Yang, P. Fournier-Viger, T. Hong, and M. Voznák. A binary PSO approach to mine high-utility itemsets. *Soft Computing*, 21(17):5103–5121, 2017.
10. J. C. Lin, L. Yang, P. Fournier-Viger, J. M. Wu, T. Hong, S. L. Wang, and J. Zhan. Mining high-utility itemsets based on particle swarm optimization. *Engineering Applications of Artificial Intelligence*, 55:320–330, 2016.
11. Y. Liu, W. Liao, and A. N. Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 689–695. Springer, 2005.

12. M. S. Nawaz, M. I. Lali, and M. A. Pasha. Formal verification of crossover operator in genetic algorithms using prototype verification system (PVS). In *9th Intl. Conference on Emerging Technologies*, pages 1–6. IEEE, 2013.
13. M. S. Nawaz and M. Sun. A formal design model for genetic algorithms operators and its encoding in PVS. In *2nd Intl. Conference on Big Data and Internet of Things*, pages 186–190. ACM.
14. W. Song and C. Huang. Discovering high utility itemsets based on the artificial bee colony algorithm. In *22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 3–14. Springer, 2018.
15. W. Song and C. Huang. Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework. *IEEE Access*, 6:19568–19582, 2018.
16. W. Song and C. Huang. Mining high average-utility itemsets based on particle swarm optimization. *Data Science and Pattern Recognition*, 4(2):19–32, 2020.
17. T. Truong, A. Tran, H. Duong, B. Le, and P. Fournier-Viger. EHUSM: Mining high utility sequences with a pessimistic utility model. *Data Science and Pattern Recognition*, 4(2):65–83, 2020.
18. J. M. T. Wu, J. Zhan, and J. C. W. Lin. An ACO-based approach to mine high-utility itemsets. *Knowledge Based Systems*, 116:102–113, 2017.
19. H. Yao and H. J. Hamilton. Mining itemset utilities from transaction databases. *Data and Knowledge Engineering*, 59(3):603–626, 2006.
20. U. Yun, D. Kim, E. Yoon, and H. Fujita. Damped window based high average utility pattern mining over data streams. *Knowledge-Based Systems*, 144:188–205, 2018.
21. Q. Zhang, W. Fang, J. Sun, and Q. Wang. Improved genetic algorithm for high-utility itemset mining. *IEEE Access*, 7:176799–176813, 2019.