Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases

Djenouri, Youcef; Djenouri, Djamel; Belhadi, Asma; Fournier-Viger, Philippe; Chun-Wei Lin, Jerry; Bendjoudi, Ahcene

Published in: Information Sciences

DOI: 10.1016/j.ins.2018.06.060

Publication date: 2019

Document version: Accepted manuscript

Citation for published version (APA):

Terms of use
This work is brought to you by the University of Southern Denmark.
Unless otherwise specified it has been shared according to the terms for self-archiving.
If no other license is stated, these terms apply:
• You may download this work for personal use only.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying this open access version

If you believe that this document breaches copyright please contact us providing details and we will investigate your claim.
Please direct all enquiries to puresupport@bib.sdu.dk
Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases

Yousef Djenouri, Djamel Djenouri, Asma Belhadi, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Ahcene Bendjoudi

PII: S0020-0255(18)30508-5
DOI: 10.1016/j.ins.2018.06.060
Reference: INS 13753

To appear in: Information Sciences

Received date: 24 January 2018
Revised date: 19 June 2018
Accepted date: 29 June 2018

Please cite this article as: Youcef Djenouri, Djamel Djenouri, Asma Belhadi, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Ahcene Bendjoudi, Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases, Information Sciences (2018), doi: 10.1016/j.ins.2018.06.060

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases

Youcef Djenouri\(^a\), Djamel Djenouri\(^b\), Asma Belhadi\(^c\), Philippe Fournier-Viger\(^d\), Jerry Chun-Wei Lin\(^e\), Ahcene Bendjoudi\(^b\)

\(^a\)Dept. of Mathematics and Computer Science, Southern Denmark University, Odense, Denmark
\(^b\)CERIST Research Center, Algiers, Algeria
\(^c\)RIMA Lab, USTHB, Algiers, Algeria
\(^d\)Harbin Institute of Technology (Shenzhen), School of Humanities and Social Sciences, Shenzhen, China
\(^e\)Department of Computing, Mathematics, and Physics Western Norway University of Applied Sciences (HVL), Bergen, Norway

Abstract
This paper investigates the use of GPU (Graphics Processing Unit) in improving the bees swarm optimization metaheuristic performance for solving the association rule mining problem. Although this metaheuristic proved its effectiveness, it requires huge computational resource when considering big databases for mining. To overcome this limitation, we develop in this paper a GPU-based Bees Swarm Optimization Miner (GBSO-Miner) where the GPU is used as a co-processor to compute the CPU-time intensive steps of the algorithm. Unlike state-of-the-art GPU-based ARM methods, all BSO steps including the determination of search area, the local search, the evaluation, and the dancing are performed on GPU. A mapping method between the data input of each task and the GPU blocks/threads is developed. To demonstrate the effectiveness of the GBSO-Miner framework, intensive experiments have been carried out. The results show that GBSO-Miner outperforms the baseline methods of the literature (GPApiroi, MEGPU, and Dmine) using big textual and graph databases. The results reveal that GBSO-Miner is up to 800 times faster than an optimized CPU-implementation.

Keywords: GPU, bees swarm optimization, association rules mining, big databases.
1. Introduction

ARM (Association Rule Mining) is a data mining technique that aims at discovering hidden and frequent patterns from a transactional databases. ARM is widely used in many applications such as Business Intelligence[13, 47], Constraint Programming [20, 21], and Information Retrieval [14, 12], where it is applied as pre-processing step to extract relationships between the input data. The generated rules helps to bias the solving process of such applications. For instance, considering the information retrieval problem, the collection of documents is transformed to the transactional database, where each document is considered as a transaction, and each term is considered as an item. In this context, ARM allows to study the different relationships between the terms of the documents. For instance, if the association rule: Information ⇒ Sciences is relevant, there is a higher independence between the terms Information and Sciences. Therefore, if the user’s looks for documents related to Information, it will be probably useful to also return documents related to Sciences. A novel term Big data mining [45] is an emerging topic that engulf sophisticated technologies and methods needed to deal with large amount of data, e.g., Google is already dealing with several Petabytes of data [29].

Approaches to ARM problem vary from complete to incomplete solutions. Complete solutions aim to extract all association rules but require huge computing resources and then CPU-time consuming[35]. These algorithms usually employed search tree to explore the association rules space such as Apriori[4], Fpgrowth [28], and SSFIM[17]. Incomplete ones aim to explore intelligently the association rule space using the swarm intelligence and mutli-objective approaches [23, 2, 1]. Without losing generality, we can cite [38, 33] for ACO (Ant Colony Optimization), [34, 41, 9, 3] for PSO (Particle Swarm Optimization), [19] for BSO (Bees Swarm Optimization). The problem of these approaches consists on the solutions’ quality represented by the percentage of the association rules. In our recent works [16], several extensions have been proposed by improving the way of exploring the rules space. Although BSO gives good results when dealing with small and medium databases, this algorithm remains inefficient in terms of runtime when considering large databases. According to [18], BSO spends up to 15days to generate only half million of satisfied rules. Parallelization is one of the most effective tools that allows to address this limitation. The use of large scale parallelism provided by the computational grids and clusters with massively parallel processors is very efficient, but those resources are excessively expensive and not easily available and accessible to users. Graphics Processing Units
(GPU accelerators) have emerged as a new support for massively parallel computing. GPUs have been initially designed for video games and entertainment, but since the year 2007, they have been used more and more as an efficient computing tool for conceiving software applications [36]. GPUs are many-core co-processor devices that provide a hierarchy of memories with different sizes and access latencies and enable a highly multi-threaded environment. A typical GPU framework is composed of two components: i) the CPU host that can be a desktop computer or server equipped with a general purpose processor and a main memory, and ii) The GPU device used as a co-processor composed of several streaming multiprocessors, each of which composed of several cores where thousands of threads run in parallel. The threads are logically organized into blocks sharing a memory. All blocks of threads have also access to constant and global memories. Physically, the threads of a block are organized into warps of 32, 64 or 128 threads.

Several parallel approaches have been developed in the literature to deal with ARM. The most effective ones are those based on Graphics Processing Units (GPUs) parallelization [18, 15]. All previous GPU-based bees swarm optimization approaches have attempted to improve ARM algorithms by only performing the evaluation step on GPU. The intuition behind this is that this operation is the most CPU time expensive. However, for big databases, diversification and intensification may also be time consuming as well. This paper investigates several GPU parallel strategies on the bees swarm optimization to improve the performance of mining big databases in terms of execution time.

This paper features four main contributions. First, we propose GBSO-Miner approach that benefits from the computing capabilities of GPUs to speedup the mining of big databases using the bees swarm optimization. Second, Contrary to state-of-the-art GPU-based ARM approaches, intensive steps of BSO (search area, local search and evaluation steps) are completely performed on GPU while reducing considerably the threads divergence. This is by proposing an improved mapping between the threads and data. Finally, we implement and test GBSO-Miner on different categories of databases including static, real small, large and big databases, and we compare GBSO-Miner with the baseline methods (MEGPU, GPApriori, and Dmine). The results are very convincing in terms of speedup, computational time, and rule’s quality.

The remainder of the paper is organized as follows: Section II reviews recent sequential and parallel GPU-based swarm intelligence ARM algorithms. Section III introduces preliminaries related to the ARM problem and the swarm intelligence approaches. Section IV describes the proposed GBSO-Miner algorithm. Then, Section V presents a performance evaluation. Finally, Section VI draws a
2. Related Work

In the last twenty-five years, many ARM algorithms have been proposed. This section reviews swarm intelligence and GPU-based ARM approaches.

2.1. Swarm Intelligence-based ARM Approaches

The first approach using ACO [38] for ARM is proposed in [32], which combines clustering and ACO. An extension of this approach for continuous domains is developed in [33]. In [37], the ant programming is used (a variation of ACO algorithm that integrates the genetic programming). Each ant constructs recursively the candidate association rules guided by a context-free grammar (CFG) that limits the search space and ensures the creation of high quality association rules. The main drawback of ACO-based approaches is the runtime performance that is due to the graph management issue.

PSO [31] has been widely applied for solving ARM problem. In [34], the authors propose a first particle swarm optimization approach where the neighborhood space is found by moving the front and back points of each particle. This algorithm favors the intensification search with respect to the diversity search. In [41], a BPSO (Binary Particle Swarm Optimization) algorithm is proposed to extract top $K$ frequent itemsets, where a binary representation is used to accelerate the fitness computing. The algorithm runs on, $K$, iterations. Vahid et al. [9] propose a multiobjective particle swarm optimization for extracting relevant association rules in numerical data. This approach allows to generate diverse and more informative pattern describing numerical data. SET-PSO (SET Particle Swarm Optimization) [3] is developed to mine both positive and negative association rules in large transactional database. First, the encoding step is separately applied to the positive and negative rules using statistical correlation among the transactional database. The set-based PSO search procedure is then established to each positive and negative candidate rules, in order to find the best association rules.

More recently, BSO is intensively applied for improving the association rules mining process. In [19], the search area of the bees is first determined, then each region is explored by one bee to find relevant association rules. At each iteration, the bees communicate using a dance table to converge to the best set of rules. Moreover, three strategies for determination of region of each bee have been investigated. They allow better diversification criterion, but they still have
the problem of the intensification. Two works that explores this problem are sug-
gested in [16]. In the first work, the authors propose a hybrid tabu search with bees
swarm optimization, where each bee explores its region using the tabu search ap-
proach. Whereas, the second work propose several strategies that allow the best
exploration of the regions and ensures a balance between intensification and di-
versification mechanisms.

Two works that introduce the bat metaheuristic developed in [30] and [44].
In the first work, a group of bats attempt to discover relevant rules in the same
region. Different strategies are proposed to select the best relevant rules among
the overall bats. The second work proposes multi-objective binary bat algorithm
based on Pareto for association rule mining. This algorithm is independent of
minimum support and minimum confidence. To evaluate the generated rules, the
authors develop a new method aims to discover interesting rules without favoring
or excluding any measure.

Three other works that use the recent metaheuristics (Penguins Search Opti-
mization, gravity search and cuckoo search) are proposed for discovering relevant
rules. In [24], the penguins search optimization [25] is proposed. Each group
of penguins interacts to construct the candidate rules according to the oxygens
reserve value. The whole swarm of penguins is then communicated to extract the
best rules. Each group of penguins still repeat this process until the oxygen re-
serve value is positive. The oxygen reserve value is updated if the resulted rules
are unsatisfied. In [42], inspired by newtonian gravity and the law of the motions,
the authors propose ARMMGSA (Association Rule Mining using Mass Gravity
and Simulated Annealing). Each rule is modeled as a mass. All masses attract
each other using the law of the motion. Then, at each iteration, the $k$-heaviest
masses are selected to influence the new masses in the next iteration by applying
force to them. The ARMMGSA algorithm generates few rules compared to other
evolutionary algorithms.

2.2. GPU-based ARM Approaches

Wenbin et al. [22] proposed a parallel version of Apriori algorithm named PBI
(Pure Bitmap Implementation). The set of transactions and the set of association
rules are represented using an efficient bitmap data structure. The rule structure
is a binary matrix with $n$ rules and $m$ items. The element located to $i^{th}$ row and
$j^{th}$ columns is set to 1 if the association rule $i$ contains the item $j$, 0, otherwise.
The transaction structure is a binary matrix with $n$ rules and $m$ transactions. The
element located to $i^{th}$ row and $j^{th}$ columns is set to 1, if the transaction $j$ contains
the rule \( i, 0 \), otherwise. The results revealed that PBI implementation is up to two orders of magnitude faster than optimized CPU-based implementation.

Zhou et al. [49] developed GPU-FPM (GPU for Frequent Pattern Mining). GPU-FPM is based on Apriori-like algorithm and uses a vertical representation of the transactions to deal with GPU memory management issue. Therefore, a compact data structure is designed and implemented to store the transactions and the generated rules. It builds MemPack structure to store different types of data such as int, float, customized structure, class, etc. The CLProgram class was also designed to launch with arbitrary number of threads, and GPU blocks. The results revealed that GPU-FPM is up to 16 orders of magnitude faster than CPU implementation using 1024 threads.

Zhang et al. [48] presented GPApriori algorithm. It used two powerful data structures for fine-grain parallelization of the support and the confidence computing. A static bitset memory structure is used to represent the set of transactions. The support and the confidence computing is based on complete intersection procedure. Thus, candidates are sent from CPU main memory to GPU global memory. The GPU calculates the support and the confidence values by performing bitwise intersections on the static bitset structure, and the resulted values are then sent back to the CPU main memory. The results revealed that GPApriori is up to 100 orders of magnitude faster than the serial Apriori algorithm, for different minimum support and confidence values.

Wang et al. [48] developed a new GPU-based Eclat algorithm that uses micron’s automata processor. It is a hardware implementation of non-deterministic finite automata. It is used for speedup the support and confidence calculation. At each pass of the algorithm, association rules are submitted to the automata processor for computing the frequency of both antecedent and consequent parts of the rules. Moreover, an efficient optimization strategy is applied to maintain automata reconfiguration for dealing with variable-size rules. The results showed that this algorithm is up to 100 orders of magnitude faster than the serial eclat algorithm using Accidents database and with minimum support set to 5%.

Gu et al. [27] suggested the BitQApriori algorithm that simplifies the process of candidate generation and support counting. Instead of generating \( k \)-candidates by combining two (\( k-1 \))-frequent itemsets, this algorithm joined the 1-frequent itemsets and the (\( k-1 \))-frequent itemsets. The bitset structure is used to store identifications of transactions that correspond to each candidate. Therefore, support and confidence counting can be implemented by using boolean operators, instead of scanning database. The speedup of this algorithm reached 16, and the throughput of the hardware design reached 217 MB for large databases.
Andrian et al. [39] implemented on FPGA (Field-Programmable Gate Array) accelerator for generate-and-test mining algorithms. An optimized support and confidence counting unit is first designed on FPGA. A low latency data delivery mechanism is then employed to maximize the FPGA efficiency. Several ARM implementations (Apriori, Eclat, and Fpgrowth) have been used to validate the proposed framework. The results showed that the Borgelt Eclat implementation outperforms the two others algorithms in terms of computational time.

Chon et al. [11] presented GMiner algorithm for large scale association rule mining. The mining process is established in in a counterintuitive way. Thus, the traversal from the first level strategy is used to generate and evaluate the candidate association rules. Moreover, the input database is divided into blocks and streams them to GPUs. This block-based streaming approach allows to solve the workload skewness problem. This approach achieved a speedup=160 for big Webdocs databases.

Some research efforts have been recently explored to introduce bio-inspired approaches for GPU-based ARM algorithms. Djenouri et al. [18] developed an intelligent mapping between the GPU threads and cluster workers is done to progress the speedup of the bees swarm optimization. The set of initial reference solutions are sent to the cluster workers, the workers explores each region of the bees and send the solutions found to the GPU. The GPU blocks then evaluates each solution in parallel and sent back the fitness computing to the workers. This process is repeated for all generated solutions.

From this succinct related work it follows that: i) The bees swarm optimization improves the computational time of the ARM process, however, they still high time consuming when dealing with big transactional databases, and ii) All the GPU-based approaches have attempted to improve ARM algorithms by only setting the evaluation step on GPU. The intuition behind this is that this operation is the most CPU time expensive. However, for big databases, diversification and intensification may also be time consuming as well. The main contribution of this work was to investigate several GPU parallelism strategies on the bees swarm optimization for improving the runtime performance of mining big databases.

3. Preliminaries

This section briefly introduces the main concepts related to association rule mining and swarm intelligence computing. Table1 Lists all the acronyms used in this paper.
Table 1: Acronyms list

<table>
<thead>
<tr>
<th>nouns</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>Association Rule Mining</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>BSO</td>
<td>Bees Swarm Optimization</td>
</tr>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>ME GPU</td>
<td>Multiple Evaluation on GPU</td>
</tr>
<tr>
<td>GPA priori</td>
<td>GPU-Apriori</td>
</tr>
<tr>
<td>Dmine</td>
<td>Distributed-mine</td>
</tr>
<tr>
<td>PSO-ARM</td>
<td>Particle Swarm Optimization for Association Rule Mining</td>
</tr>
<tr>
<td>BSO-ARM</td>
<td>Bees Swarm Optimization for Association Rule Mining</td>
</tr>
<tr>
<td>BPSO</td>
<td>Binary Particle Swarm Optimization</td>
</tr>
<tr>
<td>SET-PSO</td>
<td>SET Particle Swarm Optimization</td>
</tr>
<tr>
<td>ARM MSGSA</td>
<td>Association Rule Mining using Mass Gravity and Simulated Annealing</td>
</tr>
<tr>
<td>PBI</td>
<td>Pure Bitmap Implementation</td>
</tr>
<tr>
<td>GPU-FPM</td>
<td>GPU for Frequent Pattern Mining</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>MTD</td>
<td>Minimizing Threads Divergence</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
</tbody>
</table>

3.1. Association Rule Mining

**Definition 1 (Transactional Database).** Let \( T \) be a transactional database, defined as a set of \( m \) transactions, \( \{T_1, T_2, \ldots, T_m\} \), and \( I \) be a set of \( n \) different items \( \{I_1, I_2, \ldots, I_n\} \). A transaction \( T_i \in T \) is composed by the set of items in \( I \), and we note \( T_i \subset I \)

**Definition 2 (Association Rule).** We define association rule \( r \) as \( X \rightarrow Y \), where \( X \subset I \), \( Y \subset I \) and \( X \cap Y = \emptyset \). \( X \) and \( Y \) are the antecedent and the consequent part of the rule \( r \), respectively. The interpretation of a rule \( r \) is that if the subset \( X \) appears in a transaction, it is likely to co-occur with the subset \( Y \) according to some interestingness measure.

**Definition 3 (Support).** The support of the rule \( r \) represented by \( X \rightarrow Y \) is defined by 

\[
\text{Support}(r) = \frac{|X \cup Y|_T}{|T|}
\]

Note that \( |X \cup Y|_T \) is the number of transactions in \( T \) that contains both \( X \) and \( Y \).

**Definition 4 (Confidence).** The confidence of the rule \( r \) represented by \( X \rightarrow Y \) is defined by 

\[
\text{Confidence}(r) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}
\]
Table 2: A Transactional Database

<table>
<thead>
<tr>
<th>Transactions</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₈</td>
<td>A</td>
</tr>
<tr>
<td>T₂</td>
<td>A</td>
</tr>
<tr>
<td>T₃</td>
<td>A</td>
</tr>
<tr>
<td>T₄</td>
<td>A</td>
</tr>
</tbody>
</table>

Definition 5 (ARM Problem). The ARM problem consists of extracting all rules from a transactional database $T$ for user-defined minsup and minconf thresholds such as

$$\text{Support}(r) \geq \text{minsup}$$

and

$$\text{Confidence}(r) \geq \text{minconf}$$

Example 1. For instance, consider the transactional database presented in Table 2, which contains four transactions $\{T₁, T₂, T₃, T₄\}$ and four items $\{A, B, C, D\}$. To determine the support and the confidence of the rule $A \rightarrow D$, the number of occurrences of the subsets $\{A\}$ and $\{A \cup D\}$ are computed. It is found that $A$ appears in four transactions and that $A \cup D$ appears in two transactions. Hence, Support($A$) is 1, and Support($A \cup D$) is 1/2. The confidence of the rule $A \rightarrow D$ is thus 1/2. If the minimum support and the minimum confidence are set to 1/2, then the association rules extracted are $\{A \Rightarrow B, A \Rightarrow C, A \Rightarrow D\}$.

3.2. Swarm Intelligence

This section introduces the application of swarm intelligence for solving ARM problem. We consider two versions, i) PSO-ARM[34] (Particle Swarm Optimization for Association Rule Mining), and iii) BSO-ARM[19] (Bees Swarm Optimization for Association Rule Mining). In both versions, an association rule is considered as a vector of $n$ elements. The $i^{th}$ element is set to, 1, if the $i^{th}$ item belongs to the antecedent part of the rule, to 2, if the $i^{th}$ item belongs to the consequent part of the rule, and to 0 otherwise. For example, let us consider five items $I = \{a, b, c, d, e\}$. The rule $b \rightarrow de$ is represented by the vector $\{0, 2, 0, 1, 1\}$, whereas $a \rightarrow e$ by $\{2, 0, 0, 0, 1\}$.

In PSO-ARM[34], each particle of the swarms represents one rule, positioned randomly in the rules space. At each iteration $t$, a particle $p_i$ explores the rules space using the position $x_i$ and the velocity $v_i$, which are calculated using Eq(1) and Eq(2), respectively.
Figure 1: Runtime (Seconds) and the percentage of satisfied rules (%) of the existing bio-inspired approaches for solving ARM problem using standard databases

\[ x_i(t) = x_i(t-1) + v_i(t) \]  

\[ v_i(t) = W \times x_i(t-1) + C_1 \times (x_{pi} - x_i) + C_2 \times (x_{gi} - x_i) \]  

In Eq. 1 and 2, \( x_{pi} \) is the position of the best (highest) support and the highest confidence of the relevant association rule that \( x_i \) has observed. \( x_{gi} \) represents the best particle of swarms, i.e., the highest support and the highest confidence individual rule observed thus far by all particles. The weight, \( W \), defines the inertia of a particle and controls the trade-off between global and local experience. The parameters, \( C_1 \) and \( C_2 \), control the relative importance of the individual and global the best particles.

In BSO-ARM [19], the regions of the bees are first determined using the reference solution, and a parameter \( flip \). The \( i^{th} \) region is obtained by successfully changing from \( S_{ref} \) the bits: \{ \((1 \times Flip) + i, (2 \times Flip) + i, (3 \times Flip) + i, \ldots n - i \) \}. Each bee is then assigned to a region in order to explore it using the local search process by changing only one bit of the current solution at a time. Finally, the communication between bees is performed using a specific structure called "Dance Table". This is to elect the best solutions that will be used for the next
4. Proposed Methodology

This proposed approach, GBSO-Miner, is introduced in this section. Figure 2 presents the overall framework of GBSO-Miner. The reference rule is first sent to the GPU host and stored in the global memory. It is then submitted to the shared memory of each block. The determination of regions is then done in parallel on GPU blocks, where each region is assigned to one block. Each block explores one region using the local search process. The generated rules are evaluated in parallel, where each block evaluates the rules of its region. The rules are then put in the global memory, and the best rule becomes the reference rule for the next iteration. This process should be repeated in GPU until the maximum number of iterations is reached. In the following, the main components of the GBSO-Miner are discussed.
4.1. Determination of Search Area

In GPU-based BSO-ARM algorithm, each bee can explore independently its own region. Thus, to take advantage of the whole blocks of GPU, each block can be associated to one bee for exploring its own region. To do this, the determination of search area may be divided into several fragments, each one creating one region. That is, the threads of the block, \(i\), contribute to the determination of the region relative to the \(i^{th}\) bee. Every block will modify the reference solution according to a considered strategy. Consequently, the number of threads will be proportional to, \(n\), the number of bits of the reference solution. The following describes the three GPU parallelization strategies of the search area determination presented in [19].

4.1.1. Modulo Strategy on GPU

In this strategy, Each bee, \(k\), builds its own search area by changing successively in the solution, \(S_{ref}\), the bits, \(k + i \times Flip\), where \(i\) varies from 0 to \(n - 1\), and \(Flip\) is a predefined parameter. This strategy can be used iff the number of bees is more than \(\frac{n}{Flip}\). If the distance between solutions equals the number of different bits, then the distance between the bees and the reference solution equals \(\frac{n}{Flip}\).

**Definition 6.** The parallel version of this strategy is defined as follows: One thread of every block updates one bit of the reference rule, while the remaining threads of the same block copy the remaining bits of the reference rule. That is, the thread \(ID = (i - 1) \times Flip + 1\) of the \(i^{th}\) block updates the bit \(ID = (i - 1) \times Flip + 1\) of the reference rule for generating the \(i^{th}\) region.

The correctness of def. 6 is presented below.

**Proposition 1.** The Modulo strategy on GPU is correct with respect to the same definition for a single-processor architecture. Formally speaking, let denote by \(P_{i}\) the following proposition: the \(i^{th}\) bee modifies \(((i - 1) \times Flip) + 1)^{th}\) bit of the Reference Solution.

**Proof 1.** We follow the recurrence principle to prove the previous proposition as follows:

1. **Initialization:** The proposition is true for \(i = 1\).
2. **Implication** Assume that $P_i$ is true for $i = 1, 2, \ldots, n$ and prove that $P_{n+1}$ is true. Since $P_n$ is true then the $n^{th}$ bee updates the bit $((n-1) \times \text{Flip}) + 1$. In Modulo strategy, at each step one jump of Flip value is performed to obtain the next bee. Consequently, the $n + 1^{th}$ bee updates $((n - 1) \times \text{Flip}) + 1 + \text{Flip}$. So it updates the bit $((n \times \text{Flip}) + 1)$.

**Example 2.** Assume $S_{ref}$ reference rule of 7 items, Flip set to 1 and the number of bees set to 3. The Modulo strategy derived the regions of the bees as

1. The first bee modifies the bit 1 that corresponds to $((1 - 1) \times \text{Flip}) + 1$.
2. The second bee modifies the bit 2 that corresponds to $((2 - 1) \times \text{Flip}) + 1$.
3. The third bee modifies the bit 3 that corresponds to $((3 - 1) \times \text{Flip}) + 1$.

4.1.2. **Next Strategy on GPU**

This strategy uses a predefined number, Flip, where each bee, $k$, changes the first Flip bits in $S_{ref}$, starting from the bit $k$. For example, if Flip = 3, the bee 0 changes the bit 0, 1 and 2 and the bee 2 changes the bits 2, 3, 4, and so on. The acronym Next means that at each step, every bee, $k$, changes the bit, $k - 1$, and the next Flip − 1 bits. The distance between these solutions and $S_{ref}$ equals to Flip.

**Definition 7.** The parallel version of this strategy is defined as follows: the Flip threads of each block modify one bit of $S_{ref}$ and the remaining threads of the same block copy the remaining bits of $S_{ref}$. Indeed, the threads $ID \in [i, ((\text{Flip} + i) - 1)]$, where $i$ is the identity of the $i^{th}$ block, modify the bits assigned to them.

The correctness of def. 7 is presented below.

**Proposition 2.** The Next strategy on GPU is correct with respect to the same definition used on a single-processor architecture. That is, in the two cases, the $i^{th}$ bee modifies the bits $[i, ((\text{Flip} + i) - 1)]$ of the Reference Solution.

**Proof 2.** Let $P_i$ be the proposition: “the $i^{th}$ bee modifies the bits $[i, ((\text{Flip} + i) - 1)]$” relative to the bee $i$. This proposition may be proved by recurrence on $i$ as follows:

1. **Initialization:** The proposition is trivially true for $i = 1$.  

13
2. **Implication** Assume that $p_i$ is true for $i = 1, 2, \ldots, n$ and prove that $p_{n+1}$ is true. ($P_n$) is true then the $n^{th}$ bee modifies the \([n, ((\text{Flip}+n)-1)]\) bits of the reference solution. In this strategy, at each step, successive modifications of Flip bits are necessary to obtain the next bee. So the $n+1^{th}$ bee modifies \([n+1, ((\text{Flip}+n)-1)+1]\) and then it modifies \([n+1, ((\text{Flip}+n+1)-1)]\) that corresponds to $P_{n+1}$.

**Example 3.** Consider the previous example, and with Flip set to 2. The Modulo strategy derived the regions of the bees as

1. The first bee modifies the bits 1 and 2 that correspond to \([1, \text{Flip}+1-1]\).
2. The second bee modifies the bits 2 and 3 that correspond to \([2, \text{Flip}+1-1]\).
3. The third bee modifies the bits 3 and 4 that correspond to \([3, \text{Flip}+3-1]\).

### 4.1.3. Syntactic Strategy on GPU

Unlike the previous strategies, this one uses a syntactical, instead of random, form to generate the solutions. For this purpose, we associate the notion of weight to each solution. Formally, the weight of a solution $S = a_0a_1a_2 \ldots a_{n-1}$ noted $W(S)$ is defined as

$$W(s) = \sum_{i=0}^{n-1} a_i$$

where $n$ is the size of the solution.

For instance, if we consider the solution $S = 0112211$ then $W(S) = 0 + 1 + 1 + 2 + 2 + 1 + 1 = 6$. Thanks to this idea, each bit $i$ can generate a solution that has a gap of a given distance with $S_{ref}$. First, the algorithm computes $W(S_{ref})$. Then, each bee $k$ changes the successive bits of $S_{ref}$ starting from the bit $k$. Each bee $k$ stops this process when it obtains a solution $s$ which satisfies the constraint $W(s) = W(S_{ref}) - \text{Distance}$ or $W(s) = W(S_{ref}) + \text{Distance}$.

**Definition 8.** The parallel version of this strategy is defined as follows: One thread of each block modifies one bit of $S_{ref}$, the remaining $n - 1$ threads of the same block copy the remaining bits of $S_{ref}$, and the last threads calculate in parallel the weight of the generated solution. This process must be repeated until the distance between the current solution and the reference solution reaches the distance parameter.

This definition is trivially correct on GPU. Indeed, each block of thread generates successively potential solutions until one solution satisfies the distance criterion. Three types of threads are considered in every block.
Algorithm 1 GPU Kernel of the Determination Area Step

1: if Strategy(Modulo) = true then
2:   idx ← blockIdx.x × blockDim.x + threadIdx.x
3:   if idx = ((blockIdx.x - 1) × Flip)+1 then
4:     Rule[blockIdx.x][idx] ← Update(Sref[idx])
5:   else
6:     Rule[blockIdx.x][idx] ← Sref[idx]
7:   end if
8: end if
9: if Strategy(Next) = true then
10:   if (idx ≥ blockIdx.x) ∧ (idx ≤ Flip+blockIdx-1) then
11:     Rule[blockIdx.x][idx] ← Update(Sref[idx])
12:   else
13:     Rule[blockIdx.x][idx] ← Sref[idx]
14:   end if
15: end if
16: if Strategy(Syntactic) = true then
17:   idx ← blockIdx.x × blockDim.x + threadIdx.x
18: while (d ≠ Distance) do
19:     r ← rand(n)
20:     if (idx = r) then
21:       Rule[blockIdx.x][idx] ← Update(Sol[idx])
22:     else
23:       if (idx=n+1) then
24:         d ← W(Rule[blockIdx.x], Sref)
25:       else
26:         Rule[blockIdx.x][idx] ← Sol[idx]
27:       end if
28:     end if
29: end if
30: end while
31: end if

1. One thread to perform the modification of one bit of the reference solution.
2. \((n - 1)\) threads realized only the copy of the remaining bit of the reference solution.
3. One thread calculates the weight of the generated solution.

The general algorithm of the determination area step is given by Algorithm 1.

4.2. Local Search Area

The aim of the intensification (SLS or TS) is to explore each region by one bee. To parallelize this operation on GPU, each block of threads is associated with one region and each thread of a block modifies one bit of the reference solution. Indeed, from \(n\) items, \((n \times 2)\) neighbors can be generated so \((n \times 2)\) should be launched. Each thread, \(j\), copies the current solution, \(Sol\), in the new array called, \(Neighbor\), and it modifies the \(i^{th}\) bit by 1 or 0, where \(i = 2 \times j\), or \(i = (2 \times j) + 1\). The pseudo GPU code of this step is given by Algorithm 2.
Algorithm 2 GPU Kernel of Local Search Step

1: idx ← blockIdx.x × blockDim.x + threadIdx.x
2: for (i ← 0; i ≤ n; i++) do
3: if (idx = i/2 ∨ idx = (i/2) + 1) then
4: Copy(Neighbor[i], Bees[blockIdx.x][i])
5: else
6: Copy(Neighbor[i], Copy(Bees[blockIdx.x][i]))
7: end if
8: end for

Algorithm 3 GPU Kernel for Evaluation Step

1: idx ← blockIdx.x × blockDim.x + threadIdx.x
2: Compare the solution Buff[blockIdx.x]
3: for i ← 0 to l transactions do
4: if Buff[blockIdx.x] ∈ t((i×blockDim.x)+idx) then
5: count[blockIdx.x][i] ← 1
6: else
7: count[blockIdx.x] ← 0
8: end if
9: end for
10: fitness(Buff[blockIdx.x] ← Sum_Reduction(count[blockIdx.x])
11: cudaMemcpy(fitness(Buff[blockIdx.x]) cudaMemcpyDeviceToHost)

4.3. Evaluation

According to our previous work in [18], the evaluation step is the most CPU-time intensive operation, which needs multiple transactional database scanning. Therefore, its evaluation on GPU will intuitively reduce the overall computation time. The evaluation step on GPU deals with multiple rules simultaneously. Every block of threads matches with one rule. Threads of the same block are launched to collaboratively calculate the fitness of a single rule. Therefore, there are as many rules as blocks. The transactions are subdivided into subsets and each subset is associated with exactly one thread. Hence, each thread calculates only its corresponding subset of rules. After that, a sum reduction is applied to aggregate the fitness value. Such a strategy attempts to benefit from the massively parallel power of GPU by launching a large number of threads per rule and to reduce the CPU/GPU communications. The general procedure of slave kernel is given by Algorithm 3.

4.4. Improvements

Let us recall that in ARM problem, a database is a set of transactions with different number of items. Thread divergence is caused by two reasons. First, each thread handles different number of items. In this case, there are threads that finish before others. Second, the comparison process of a given thread is stopped.
when it does not find a given item of the considered rule in the transaction to which it is mapped.

The number of thread divergence (TD) can be computed as a function of the number of comparisons done by the different threads as follows,

$$TD = \max\left\{\max\left\{\left|t_{(r+w)+i}\right| - \left|t_{(r+w)+j}\right|\right\} / (i, j, r) \in [1...w^3]\right\}.$$  

(3)

Where

$\left|t_{(r+w)+i}\right|$ is the size of the $(r * w) + i^{th}$ transaction assigned to the $i^{th}$ thread allocated to the $r^{th}$ warp. $w$ is the number of warps.

Furthermore, threads divergence can be computed according to the distribution of items in transactional database. Consequently, we distinguish the following two cases:

**Irregular Distribution of Items:** In this case, the transactions are highly different in size. In the worst case, we find on the same warp one transaction containing all items and another containing only a single item. For this, we can approximate threads divergence to the maximal number of items (reduced by one) when the data set is very large. That is,

$$\lim_{M \to +\infty} TD(M) = N - 1.$$  

(4)

**Regular Distribution:** Unlike the first case, here we notice a slight difference between the size of transactions. If we denote, $r_1$, the variation between transactions, then,

$$\lim_{M \to +\infty} TD(M) = r_1.$$  

(5)

To reduce threads divergence, we attempt to improve the assignment of the transactions on different blocks. The assignment of the transactions is performed according to the number of items in each transaction. Indeed, the transactions of $i$ items are assigned to the $i^{th}$ block. Consequently, the number of blocks is equal to the number of items. The pseudo algorithm of this strategy is given in Algorithm 4.

Using this strategy, the thread divergence between threads of the same warp is minimized because the threads of each block have the same number of transactions. However, the load balancing between blocks is not taken into account
Algorithm 4 MTD (Minimizing Threads Divergence) Strategy

1: for each transaction \( t \) do
2: \( nb = \text{calculate Number Of Items}(t) \)
3: assign \( t \) to the \( nb^{th} \) block
4: end for

if transactions that have the same number of items. In fact, some blocks handle a large number of transactions while others handle few transactions. Such effect degrades the performance of ARM.

Example 4. Consider the transactional database that contains five different items \( \{A, B, C, D, E\} \) and ten transactions as:

- \( t_1 \) : A, B, C.
- \( t_2 \) : A, C.
- \( t_3 \) : B, C.
- \( t_4 \) : A, B, C, D, E.
- \( t_5 \) : C, D, E.
- \( t_6 \) : D, E.
- \( t_7 \) : B, D.
- \( t_8 \) : A, E.
- \( t_9 \) : A, B, C, D, E.
- \( t_{10} \) : A, D, E.

We first compute the number of items of each transaction. An array of integer is created, its \( i^{th} \) element represents the number of items of the \( i^{th} \) transaction as:

\[
\begin{array}{ccccccccccc}
 t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\
 3 & 2 & 2 & 3 & 3 & 2 & 2 & 2 & 3 & 3 \\
\end{array}
\]

The number of blocks is equal to the number of items which is 5. Consequently, the size of each block is equal to 2.

The first transaction contains 3 items so it is assigned to the third block. The second transaction is assigned to the second block, and so on until the assignment of all transactions. The final status of the blocks are given on Fig. 3.

According to this scheme, only three from five blocks are used (block\(_2\), block\(_3\) and block\(_5\)). To improve this strategy, the two other blocks, block\(_1\) and block\(_4\), will be deallocated before launching ARM process. On each occupied block, we dispose two threads, so the block\(_2\), handles their transactions on three steps two transactions (2 transactions then 1 transaction). The block\(_3\) handles their transac-
tions on two steps and the last block on one step, which makes unbalanced blocks. If we have the same density on each group of transactions with the same number of items; this strategy ensures load balancing. For instance, if we consider the following case:

\[ t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10} \]

1 1 2 2 3 3 4 4 5 5

Each block handles two transactions simultaneously.

Complexity

Proposition 3. The complexity of BR strategy is \( O((N + 1) \times M) \).

Proof 3. In Algorithm 4, the number of items of each transaction, \( t \), is first calculated, which requires in the worst case \( O(N) \) steps, where \( N \) is the number of all items. After that, we assign \( t \) to the \( nb^{th} \) block where \( nb \) is the number of items of \( t \). This operation is performed in one pass, so its complexity is \( O(1) \). The algorithm is repeated for all transactions. If the number of transactions is equal to \( M \), then, The complexity of this strategy is \( O((N + 1) \times M) \).

4.5. Theoretical Analysis of GBSO-Miner

To design any GPU-based parallel approach, two main challenges should be respected. i) CPU-GPU Communication, where the optimization of data transfer between the CPU and GPU is necessary, and ii) threads divergence. The second challenge is due to the fact that GPU follows SIMD model, i.e., threads located at the same wrap (32 threads in almost all GPU versions) should execute the same instruction with different data. The aim here is to distribute workload equally among all the GPU threads by minimizing the threads divergence. In this section,
a theoretical study of the GBSO-Miner is presented by analyzing the challenges related to the GPU architecture. Moreover, a comparison between a theoretical complexity of GBSO-Miner and the sequential version is also made.

1. **CPU-GPU communication**

   The CPU/GPU communication only consists of transferring the reference rule to the GPU. Therefore, the CPU/GPU communication cost is $N$ Bytes.

2. **Threads divergence**

   **Proposition 4.** The total number thread divergence count of the proposed approach is $(N - 1)$.

   **Proof 4.** All tasks are performed on a GPU, so the total number of threads divergence is computed by summing the thread divergence count caused by the determination of regions, the local search process, and the evaluation process. For the determination of regions, each thread has to generate one region for a bee. All threads of a given block have to execute the same instruction. Consequently, there is no thread divergence between threads of the same block. For the local search process, each thread of the given block generates one solution. All threads of such block have to execute the same instruction at the same time. Hence, there is no thread divergence between threads of the same block. For the evaluation step, the transactions are usually different in terms of number of items. To evaluate a single rule on GPU, the different threads have to scan all its items and compares them to the transaction it is mapped with. The comparison process of a thread is stopped when it does not find a given item of the considered rule in the transaction it checks. Thread divergence of this step can be computed according to Equation 4, which is equal to $N - 1$. Consequently, the threads divergence of GBSO-Miner is $N - 1$.

3. **Time complexity**

   The time complexity of any metaheuristic-based approach depends on the empirical parameters used in the searching process. As the proposed algorithms are based on BSO, their computation complexity depends on the following parameters: $IMAX$ is the maximum number of iterations to be performed by the algorithm, $K$ the number of regions or bees in the colony, $N$ is the number of items (i.e. solution size), and $M$ is the number of transactions. The complexity cost of the sequential version is in $O(IMAX \times K \times N \times M)$ [19].

   **Proposition 5.** The complexity of GBSO-Miner is $O(IMAX)$.
Proof 5. With GBSO-Miner, all BSO tasks are performed on GPU in IMAX iterations. The complexity is hence $O(IMAX)$.

We can see that GBSO-Miner reduces the time complexity of the sequential version published in [19].

5. Performance Evaluation

To validate the usefulness of the GBSO-Miner algorithm, intensive experiments have been carried out using small, large and big databases. In this section, we first describe the databases used in the experiments. Three tests are then established: (i) Using syntactic database, the GBSO-Miner has been tested by varying with the number of transactions, the number of items, the determination of the used search area, and the minimum support value, ii) Using small and large real databases, GBSO-Miner has been tested with and without employing minimizing threads divergence strategy, and (iii) the best implementation of GBSO-Miner is then compared with the state-of-the-art ARM algorithms using big databases by varying with the number of blocks and the number of threads.

5.1. Data Instances Description

We consider three sets of databases:

1. Synthetic databases generated using the IBM Quest data generator.
2. Small and large real databases available retrieved from.

Table 3 lists the number of transactions, number of items and average number of items per transactions of small and large real databases.

The IBM Quest data generator allows to generate large instances of variable size. The number of transactions can be varied from 1 million to 50 million using multiples of hundreds of thousands. Each generated transaction has at least 12 items preceded by a unique transactional ID. The largest dataset obtained is in the order of 10 Gbytes.

As far as the third set of data instances is concerned, we consider the following two real big data instances:

1. Webdocs: It is built from a collection of links among html documents avail-

\[\text{1https://sourceforge.net/projects/ibmquestdatagen/}\]
\[\text{3http://fimi.ua.ac.be/data/webdocs}\]
Table 3: Small and Large Real Databases Description

<table>
<thead>
<tr>
<th>Instance Class</th>
<th>Instance Name</th>
<th>N. of transactions</th>
<th>N. of items</th>
<th>Avg. size of transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCI Machine Learning</td>
<td>Zoo</td>
<td>102</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Lymphography</td>
<td>148</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Soybean</td>
<td>683</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Australian</td>
<td>691</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Izmir Weather</td>
<td>1461</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Segment</td>
<td>2310</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Splice</td>
<td>3190</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Nursery</td>
<td>12960</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>House 16H</td>
<td>22784</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Connect-4</td>
<td>67557</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>IBM-Quest-standard</td>
<td>100000</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Chess</td>
<td>3196</td>
<td>75</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Mushroom</td>
<td>8123</td>
<td>119</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>PumbStar</td>
<td>40385</td>
<td>7116</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>BMS-WebView-1</td>
<td>59802</td>
<td>497</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>BMS-WebView-2</td>
<td>77512</td>
<td>3348</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Korasak</td>
<td>80769</td>
<td>7116</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Retail</td>
<td>88162</td>
<td>16469</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>IBM-Artificial</td>
<td>100000</td>
<td>999</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>BMP-POS</td>
<td>515597</td>
<td>1657</td>
<td>2.5</td>
</tr>
</tbody>
</table>

able on the Web. The data instance obtained contains 1.692,082 transactions with 526,765 items. The maximal length of each transaction is about 70,000 items. The size of the database is 1.48 GB.

2. Pockec: It is obtained from the logs of the most popular social network in Slovakia. It contains 1,351,254 transactions with 102,654 items. The size of the database is 1.02 GB.

5.2. Results on Synthetic Databases

Figure 4 presents the speedup of GBSO-Miner using syntactic databases, and by setting with the strategy of determination area step. By varying with the number of transactions from 1 to 2 million, the number of items from $100K$ to $1000K$, and the minimum support from 100% to 10%, the speedup of the GBSO-Miner increases whatever the strategy used in the determination of search area step. Moreover, Modulo strategy outperforms the two other strategies (Next and Syntactic). Indeed, the speedup of GBSO-Miner(Modulo) reached 500, however, the speedup of GBSO-Miner(Next) does not exceed 470 and the speedup of GBSO-Miner(Syntactic) does not exceed 410. According to these results, the Modulo strategy is used for the remaining of the experiment.

4http://snap.stanford.edu/data/soc-pockec.html
Figure 4: Speedup of GBSO-Miner using syntactic databases with varying the number of transactions from 1 to 2 million, the number of items from 100K to 1000K, and the minimum support from 100\% to 10\%.
Table 4: Solution’s Quality in terms of Percentage of Satisfied Association Rules on Small and Large Real Databases

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>BSO Quality</th>
<th>GBSO-Miner Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoo</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Lymphography</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Soybean</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Australian</td>
<td>99</td>
<td>100</td>
</tr>
<tr>
<td>Izmir Weather</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Segment</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Splice</td>
<td>95</td>
<td>94</td>
</tr>
<tr>
<td>Nursery</td>
<td>97</td>
<td>95</td>
</tr>
<tr>
<td>House 16H</td>
<td>94</td>
<td>93</td>
</tr>
<tr>
<td>Connect-4</td>
<td>91</td>
<td>92</td>
</tr>
<tr>
<td>IBM-Quest-standard</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Chess</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Mushroom</td>
<td>95</td>
<td>93</td>
</tr>
<tr>
<td>PumbStar</td>
<td>90</td>
<td>94</td>
</tr>
<tr>
<td>BMS-WebView-1</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>BMS-WebView-2</td>
<td>88</td>
<td>91</td>
</tr>
<tr>
<td>Korzak</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>Retail</td>
<td>89</td>
<td>91</td>
</tr>
<tr>
<td>IBM-Artificial</td>
<td>90</td>
<td>88</td>
</tr>
<tr>
<td>BMP-POS</td>
<td>87</td>
<td>86</td>
</tr>
</tbody>
</table>

5.3. Results on Small and Large Real Databases

Table 4 shows the quality of the solutions in terms of percentage of satisfied association rules on small and large real databases. According to this table, we notice that the quality of the solutions of GBSO-Miner and BSO are close, and the parallel approach converges to the sequential solution for all cases. Moreover, the BSO outperforms GBSO-Miner for 7 cases, GBSO-Miner outperforms for other 8 cases, and both approaches are the same in the remaining 7 cases. These results are obtained because we follow the logic of BSO in implementing the GPU parallel version. The slight difference between both approaches are observed because of the randomized search process realized by BSO algorithm.

Table 5 presents the speedup of GBSO-Miner with and without minimizing threads divergence strategy described in Section 4.4 on small and large real databases. According to this table, we can remark that the speedup of GBSO-Miner increases by increasing the number of transactions and items. For small databases such as Zoo, Splice, and IBM-Quest-standard, the speedup does not exceed 50, however, for large databases such as BMP-POS, IBM-Artificial, and Retail, the speedup reaches 800. These results are obtained thanks to the intelligent mapping between the massively GPU threaded and the data input in all BSO tasks, including, the determination of regions, the local search and the danc-
Table 5: Speedup on Small and Large Real Databases

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>GBSO-Miner (Without MTD)</th>
<th>GBSO-Miner (With MTD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoo</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Lymphography</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>Soybean</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Australian</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Izmir Weather</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>Segment</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>Splice</td>
<td>21</td>
<td>28</td>
</tr>
<tr>
<td>Nursery</td>
<td>45</td>
<td>64</td>
</tr>
<tr>
<td>House 16H</td>
<td>53</td>
<td>76</td>
</tr>
<tr>
<td>Connect-4</td>
<td>91</td>
<td>125</td>
</tr>
<tr>
<td>IBM-Quest-standard</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>Chess</td>
<td>23</td>
<td>34</td>
</tr>
<tr>
<td>Mushroom</td>
<td>59</td>
<td>79</td>
</tr>
<tr>
<td>PumbStar</td>
<td>231</td>
<td>284</td>
</tr>
<tr>
<td>BMS-WebView-1</td>
<td>257</td>
<td>301</td>
</tr>
<tr>
<td>BMS-WebView-2</td>
<td>306</td>
<td>387</td>
</tr>
<tr>
<td>Korasak</td>
<td>320</td>
<td>402</td>
</tr>
<tr>
<td>Retail</td>
<td>341</td>
<td>419</td>
</tr>
<tr>
<td>IBM-Artificial</td>
<td>398</td>
<td>491</td>
</tr>
<tr>
<td>BMP-POS</td>
<td>687</td>
<td>803</td>
</tr>
</tbody>
</table>

The results also reveal that the improvements step reduces the overall computation time, and that the GBSO-Miner with MTD strategy is faster than the GBSO-Miner without BR strategy. This is explained by the effectiveness of load balancing of GBSO-Miner by employing the MTD strategy versus the GBSO-Miner without BR strategy. Consequently, the last experiment consists on evaluating GBSO-Miner using BR strategy on big real databases.

5.4. Results on Big Real Databases

This experiment has the goal to compare the GBSO-Miner with the most effective methods of the literature (GPApriori [48], and MEGPU[18]) using the big Webdocs database. Figures 5 present the runtime of GBSO-Miner, GPApriori and MEGPU for extracting association rules from Webdocs database. By varying the number of GPU blocks and the number of threads from 128 to 1024, the GBSO-Miner outperforms both GPApriori and MEGPU approaches for all cases. These results are obtained thanks to the intelligent mapping between the association rules and the GPU blocks for all BSO steps including determination of search area, local search, evaluation, and dancing step. In addition, the efficient strategy used to minimize threads divergence issue.

The last experiment aims to extend GBSO-Miner for dealing with big graph instances. For this, GBSO-Miner is adapted to mine frequent graphs. We compare
Figure 5: Runtime (Seconds) of GBSO-Miner and state-of-the-art ARM algorithms using Web-docs with varying the number of GPU Blocks and Threads from 128 to 1024.
Figure 6: Runtime (Seconds) of GBSO-Miner and Dmine using Pockec with varying the number of GPU Blocks and Threads from 128 to 1024
GBSO-Miner and Dmine [46], the most relevant frequent graph mining algorithm. Figures 6 present the runtime of both approaches (GBSO-Miner and Dmine) for extracting association rules from the big Pockec graph database containing 1.63 million transactions, and 269 different items. By varying the number of GPU blocks and the number of threads from 128 to 1024, it is observed that GBSO-Miner outperforms Dmine for all cases. These results are also due to the fact that GBSO-Miner well exploits GPU-parallelism in all BSO steps to explore big rule space.

5.5. Discussion

For the sake of conciseness, in the remainder of this section we discuss the main findings from the use of GPU parallelism in exploring the bees swarm optimization for mining big databases.

• The first finding of our study is that the GBSO-Miner framework outperforms the baseline approaches for mining big databases such as GPApriori, MEGPU, and Dmine.

• The second finding is that GBSO-Miner is up to 800 orders of magnitude faster than the optimized CPU implementation using large real database such as BMP-POS. These results are obtained without losing on the quality of the solutions, which GBSO-Miner converges to the CPU implementation version in all cases.

• Based on a task distribution, our approach is the first approach in the literature, that considers implementing all BSO steps in GPU.

• Being based on GPU optimization, our approach deals with one of the most important GPU challenging issues, that minimize both CPU-GPU communication and threads divergence.

• From a data mining research standpoint, our paper is an example of the application of a big data mining framework [45] on dealing with big databases.

6. Conclusion

In this paper, we deeply exploit the GPU parallelism on the bees swarm optimization for mining big databases by developing the GBSO-Miner. The determination of search area, the local search, the evaluation, and the dancing steps are performed in parallel in GPU to benefit from the computing power provided
by the massively parallel GPU threads. In addition, an intelligent strategy to minimize threads divergence is developed. To validate the GBSO-Miner performance, several experiments have been established on syntactic, small, large, and big databases. The results show that GBSO-Miner reached speedup of 800 without losing in the quality of the solutions when handling the different databases. In the big data context, GBSO-Miner outperforms GPApriroi, and MEGPU the most effective GPU-based approaches in the literature using Webdocs database by varying with the GPU parameters (number of blocks and number of threads). Moreover, when dealing with the big graph Pockec database, the results reveal the ability to extend GBSO-Miner to mining frequent graphs by outperforming the baseline DMine algorithm. As a perspective, we plan to apply GBSO-Miner in some excited applications for improving their speedup such as deep learning [10], text analysis [7, 6], image processing [8], and network analysis [43, 26, 40, 5].

References


31