Parallelism in the Generation of Concepts Through the Formal Context Object Partitioning Using the In-Close 4 Algorithm

André Alves, Luis Zarate, Henrique Freitas and Mark Song
Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica de Minas Gerais, Brazil

Keywords: FCA, Parallelism, Concept Generation, In-Close 4.

Abstract: Formal Concept Analysis (FCA) usage for information extraction has been increasingly recurrent, using algorithms already developed for this purpose. However, the computational effort involved in analyzing and extracting information can drastically increase in the case of dense and high-dimensionality databases. The main goal of this work is to present a parallel approach to solving this problem. We propose parallelizing In-Close 4 with C++ and OpenMP to generate formal concepts in multiple threads. Our approach is based on the subdivision of formal contexts into subcontexts grouped by a single set of objects. In addition, we propose a set of operations to obtain the original formal concepts from the quasi-concepts, concepts generated from the subcontexts. Our results show a speedup of up to 1.4116x with an efficiency of 70.48% for 100,000 objects, 50 attributes, and a density of 50%.

1 INTRODUCTION

The quest for knowledge has become more accessible as technology improves over time. As a result, new information search mechanisms have emerged with increasing frequency, and existing ones are becoming more efficient. The internet is the biggest provider of this data, which has generated the ambition of the scientific community to facilitate the acquisition of knowledge on this platform (Godinho et al., 2022).

Since there are huge data repositories on the internet and it would be unfeasible for a human to process them due to the need for prior knowledge to define the relevance of the information, several data mining techniques have been proposed to obtain knowledge (Domingos-Silva and Vieira, 2008). The Formal Concept Analysis (FCA) stands out among these techniques, focusing on the relationships between the elements (Carpineto and Romano, 2004).

FCA is a branch of mathematics created for data analysis through associations and dependencies of objects and attributes formally described from a real, or a synthetic dataset (Ganter and Wille, 2012).

To represent these data, the Formal Context consists of a set of attributes and objects and a set of Incidences, which consist of the pair between a subset of objects and attributes.

Through the contexts, it is possible to obtain the formal concepts and organize them hierarchically in formal lattices to discover knowledge by providing an explicit hierarchy between the concepts in the database.

FCA for data analysis is widely used in several fields of knowledge, such as psychology, artificial intelligence, biology, and linguistics (Priss, 2006; Jindal et al., 2020). The reason is due to FCA's characteristic of hierarchically representing the relationships between the datasets to be analyzed.

However, the computational effort involved in analyzing and extracting information can drastically increase in the case of dense and high-dimensionality databases. Therefore, the extraction of the concepts for those scenarios becomes unfeasible.

In this way, several techniques have emerged to reduce the complexity of the generation of these concepts. An example is parallelism, such as hybrid architectures for performance improvements in algorithms that use FCA approaches (de Moraes et al., 2016; Chunduri and Cherukuri, 2019; Novais et al., 2021).

A possible alternative to generating formal concepts is based on the subdivision of the set of objects. Then, smaller subsets can be processed in parallel threads.

Thus, the main goal of this work is to present an approach focused on reducing the In-Close 4 algorithm execution time. Our proposal was developed in C++ and OpenMP to generate formal concepts in

Alves, A., Zarate, L., Freitas, H. and Song, M.
Parallelism in the Generation of Concepts Through the Formal Context Object Partitioning Using the In-Close 4 Algorithm.
DOI: 10.5220/0011991500003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 195-202
ISBN: 978-989-758-648-4; ISSN: 2184-4992

parallel from the subdivision of formal contexts into subcontexts. Each subcontext is a set of objects and represents a thread workload. As part of the solution, a set of operations was also proposed to make it possible to obtain the original formal concepts from the quasi-concepts, i.e., concepts generated from the subcontexts.

The execution times of the In-Close 4 algorithm in its serial version and our parallel proposal were evaluated. For instance, we achieved a speedup of up to 1.4116x with an efficiency of 70.48% for 100,000 objects, 50 attributes, and a density of 50%.

Thus, our main contribution to state-of-art is a new approach for generating formal concepts using quasi-concepts, targeting parallelism exploitation. We also contribute with a parallel code using C++ and OpenMP, which is available for download[1].

This paper is organized as follows: Section 2 shows a background with some important concepts; Section 3 describes the methodology; Section 4 presents and discusses the experiments and results; Section 5 shows related work, and Section 6 discusses some threats to validity and limitations. Finally, Section 7 presents the conclusions and future works.

2 BACKGROUND

2.1 Formal Concept Analysis

Formal concept analysis is the branch of applied mathematics that studies the mathematization of concepts and their representation in a hierarchical way, called the conceptual lattice.

The first known appearance was in 1982. The author proposed considering lattice elements as concepts to establish a common language in the lattice area (Ganter et al., 2005).

In its most classical approach, better known as dyadics, a formal context is expressed as a triple $K = (G, M, I)$, which G represents a group of objects, M represents a group of attributes and I represents a binary relationship between G and M, indicating that an object $g \in G$ has an attribute $m \in M$. Table 1 represents a dyadic context with $|\boldsymbol{G}| = 8$, $|\boldsymbol{M}| = 9$ and $|\boldsymbol{I}| = 33$.

A formal concept of a formal context, expressed by $K = (G, M, I)$, is defined by a pair (A, B), where $A \subseteq G$, $B \subseteq M$. From this, we have that the pair A, B follows the conditions in which $A = B'$ and $B = A'$, with $(')$ representing a derivation operator, defined as:

Table 1: Formal dyadic context.

	needs water to live	lives in the water	lives on land	needs chlorophyll	dicotyledon	monocot	can move	has limbs	breastfeed
leech	X	X					X		
bream	X	X					X	X	
frog	X	X	X				X	X	
dog	X		X				X	X	X
weeds	X	X		X		X			
reed	X	X	X	X		X			
beans	X		X	X	X				
corn	X		X	X		X			

$$A' = \{b \in M \mid aIb \; \forall a \in G\} \tag{1}$$

$$B' = \{a \in G \mid aIb \; \forall b \in M\} \tag{2}$$

The set of objects A is defined as the *extension* of a formal concept - all objects present in A contain all attributes present in B. The set of attributes B is defined as the *intention* of a formal concept - all attributes present in B that belong to the set of objects A.

As an example from Table 1, we can visualize the pair (A, B) defined by $\{$*"frog", "reed"*$\}$, $\{$*"needs water to live", "lives in the water", "lives on land"*$\}$) as a formal concept, since $A' = B$ and $B' = A$. Table 2 shows some formal concepts related to the data in Table 1.

Table 2: Formal concepts related to Table 1.

	Extension	Intent
Concept 1	{weeds, reeds, corn}	{needs water to live, needs chlorophyll, monocots}
Concept 2	{frog, reeds}	{needs water to live, lives in the water, lives on land}
Concept 3	{leech, bream, frog, dog, weed, reed, bean, corn}	{needs water to live}
Concept 4	{bream, frog}	{needs water to live, lives in the water, can move, has limbs}
Concept 5	{bream, frog, dog}	{needs water to live, can move, has limbs}

[1] https://github.com/alvesandre/ConceptsFinder

2.2 Synthetic Formal Contexts

This work used several synthetic formal contexts to guarantee a better exploration of the algorithm functionality.

We use the *SCGaz*, a tool proposed by Rimsa et al. (Rimsa et al., 2013) to create synthetic formal contexts. The tool is capable of generating random contexts with unique objects and custom sizes, dimensions, and densities.

2.3 Concept Extraction Algorithms

There are currently several algorithms in the literature to extract the concepts from the generated formal contexts. Among them, two algorithms stand out: In-Close(Andrews, 2009) and Data-Peeler(Cerf et al., 2008).

In-Close(Andrews, 2009) uses incremental closures and matrix searches to extract all formal concepts that adhere to a formal context. It used an implicit search lexicographic approach to avoid the computational overhead for repeated closures. Closing is completed incrementally and only once per concept as it walks through the attributes. This processing continues, adding a new attribute and pruning branches until the concept is closed (Andrews and Orphanides, 2010).

Data-Peeler(Cerf et al., 2008) is an algorithm that uses in-depth searches. It works using two parameters: the current node (N) and its related stack (P). The algorithm starts from the root node and uses an empty stack. After that, the proximity property is checked, as well as a monotonic constant C defined by the user. If both conditions are met, the n-set U will be output as long as there are no more elements to enumerate or if the enumeration process continues by splitting the current node N into two new nodes.

Each node N in the enumeration tree is a pair (U, V) where U and V are two n-sets. N represents all n-sets, which contain all the elements of U and a subset of the elements of V. In other words, this is the n-set search space. The root node represents all possible n-sets.

2.4 Parallel Computing

Parallel computing (Pacheco, 2011; Robey and Zamora, 2021) is a form of computation in which several calculations are performed simultaneously, acting on the premise that large problems can usually be divided into smaller ones so that they can be solved in the same amount of time (in parallel). This technique can be used in image processing and climate modeling, among other areas (Quinn, 2003; Linden, 2008).

Therefore, using processing cores to execute multiple tasks can give us higher performance. In addition, parallel computing tools can improve computational resources, dividing a larger task into several sub-tasks and executing them simultaneously in several cores.

The *OpenMP API* (Mattson et al., 2019; Van Der Pas et al., 2017) is an alternative to exploit parallelism on shared-memory machines. Besides, it is possible to use OpenMP for heterogeneous computing, which includes Central Processing Unit (CPU) and, e.g., Graphics Processing Unit (GPU) and Intel Xeon Phi.

3 METHODOLOGY

The context used to execute some simulations presented in this work is available with the executable of the In-Close algorithm[2]. SCGaz tool was also used to create synthetic contexts used in the tests (Rimsa et al., 2013). It is also important to point out that the algorithm chosen for extracting formal concepts was In-Close, in version 4, due to the ease of handling and parameterization since it allows the customization of a minimum number of attributes and objects.

The language used to develop the algorithm used in the tests was the *C++* language, with some wrappers to execute tests automatically. The *OpenMP API* was also used to exploit multithreading parallelism. The algorithm is available on the GitHub website[3].

3.1 Thread Parallelization

For parallelization, the *OpenMP* directives were used to execute threads in parallel. The number of threads was defined by the number of divisions for each context. *OpenMP* was also used to generate synthetic formal contexts in parallel.

3.2 Generation of Quasi-Concepts and Formal Concepts

A formal context of example present in the selected algorithm for the analysis was used to generate the formal concepts used in this work. After that, the formal concepts of this context were extracted using the In-Close 4 algorithm, using the following parameters:

- Minimum number of attributes: 0

[2]https://upriss.github.io/fca/examples.html
[3]https://github.com/alvesandre/ConceptsFinder

- Minimum number of objects: 0
- Output format: JSON file (*JavaScript Object Notation*)

The minimum number of attributes and objects was chosen as the minimum possible for the algorithm to extract as many formal concepts as possible. The output format chosen was JSON due to the convenience of handling these types of files. These formal concepts were generated to validate the formal concepts generated from the quasi-concepts.

Then, the formal context was partitioned to subdivide it into subcontexts - dividing them into subsets of objects and keeping the attributes and their respective incidences.

After that, concepts were extracted from each formal subcontext. Each concept obtained is called a *quasi-concept*. This may or may not become a concept when considering the formal context as a whole. Therefore, a new thread was used for each subcontext (a subdivision of formal context) to generate quasi-concepts executed in parallel.

It is important to emphasize that the operations will always be performed between two groups of quasi-concepts and whenever we execute the operations where the number of quasi-concept groups is greater than two, the first operation will be performed between the first two groups, but the next ones will be made between the previous result and the next group.

3.3 Operations

Consider $K(G,M,I)$ a context subdivided in subcontexts $K_i(G_i,M,I_i)$, $1 \leq i \leq n$, where $G_p \cap G_q = \emptyset \; \forall \; p \neq q$ and $G_1 \cup G_2 \cup ... \cup G_n = G$. Note that G_i is a partition of G.

Assuming that (A_p,B_p) and (A_q,B_q) are quasi-concepts obtained from subcontexts K_p and K_q. Based on the quasi-concepts generated from each subcontext K_i, it is necessary to perform the following operations to obtain the final concepts:

1. If $B_p \cap B_q = \emptyset$ then the quasi-concepts are concepts.

Proof: As $B_p \cap B_q = \emptyset$ then $B'_p = A_p$ and $A'_p = B_p$ for the whole context K. The same idea applies to any (A_q,B_q).

Table 3 shows an example for context K. To simplify it, we use two subcontexts K_1 and K_2, but the same approach is easily extended to n subcontexts.

Table 3: Example of quasi-concepts that are concepts.

		a	b	c
K_1	1	X	X	
	2	X		
	3	X	X	
K_2	4			X
	5			X
	6			X

It is possible to see that ({1, 2, 3}, {a}) and ({1, 3}, {a, b}) are quasi-concepts obtained from subcontext K_1. Also, ({4, 5, 6}, {c}) is a quasi-concept obtained from subcontext K_2.

As {a} ∩ {c} = ∅, then ({1, 2, 3}, {a}) is also a concept for the whole context K since {a}' = {1, 2, 3}, and {1, 2, 3}' = {a} considering all context K.

The same idea applies for ({1, 3}, {a, b}). Since {a, b} ∩ {c} = ∅, then ({1, 3}, {a, b}) is also a concept for the whole context K since {a, b}' = {1, 3}, and {1, 3}' = {a, b} considering all context K.

Thus, ({4, 5, 6}, {c}) is also a concept for the same reason.

2. If $B_p \cap B_q \neq \emptyset$ then the set of attributes $B_p \cap B_q$ is common to all set of objects A_p and A_q, so there is a formal concept with objects $A_p \cup A_q$ and attributes $B_p \cap B_q$.

Proof: Consider X = $A_p \cup A_q$ and Y = $B_p \cap B_q$. If $B_p \cap B_q \neq \emptyset \implies Y' = X$. But $X' = Y$, so ($A_p \cup A_q$, $B_p \cap B_q$) is certainly a concept of K.

Table 4 shows an example of 2 subcontexts of a context K. The same approach is easily extended to n subcontexts.

Table 4: Example for intersection.

		a	b	c
K_1	1	X	X	X
	2	X	X	
	3	X	X	
K_2	4	X	X	X
	5	X		X
	6	X		

In the example, it can be seen that from the subcontext K_1, the quasi-concept $(\{1\}, \{a,b,c\})$ is obtained. From the second subcontext K_2, it is possible to obtain the quasi-concept $(\{4\},\{a,b,c\})$.

We can infer a concept from these two quasi-concepts, since both share the set of attributes {a,b,c}. Therefore, it is possible to obtain the formal concept $(\{1,4\},\{a,b,c\})$, since the intersection of the sets of

intentions resulted in $\{a,b,c\}$ and the union of the extension set $\{1\} \cup \{4\}$ is equal to $\{1,4\}$.

It is important to note that, from the subcontext K_1, the quasi-concept $(\{1,2,3\}, \{a,b\})$ is obtained. From the second subcontext K_2, it is possible to obtain the quasi-concept $(\{4,5,6\},\{a\})$.

We can infer a concept from these two quasi-concepts since both share the set of attributes {a}. Therefore, it is possible to obtain the formal concept $(\{1,2,3,4,5,6\},\{a\})$, since the intersection of the sets of intentions resulted in $\{a\}$ and the union of the extension set $\{1,2,3\} \cup \{4, 5, 6\}$ is equal to $\{1,2,3,4,5,6\}$.

3. If $B_p \subseteq B_q$ then the quasi-concepts are subconcepts of a concept in context K. To perform this action, check if one of the attribute sets is a subset and if the set of objects of a quasi-concept is a subset of objects of an extracted concept. Table 5 shows an example.

Table 5: Example for quasi-concepts that are not concepts.

		a	b	c
K_1	1	X		X
	2	X	X	X
	3	X		X
K_2	4	X	X	X
	5	X		X
	6	X		X

It can be observed that in the first subcontext K_1 there is a quasi-concept $(\{1,2,3\},\{a,c\})$ and in the second subcontext K_2 there is another quasi-concept $(\{4,5,6\},\{a,c\})$. However, both are not concepts, as they share the intention $\{a,c\}$ that is included in a concept $(\{1,2,3,4,5,6,\},\{a,c\})$ obtained from rule 2.

4. Supremum and Infimum

To calculate the supremum and infimum, when they are not generated as quasi-concepts, it is necessary to define the infimum as the set of objects that share all attributes and the supremum as the set of attributes that share all objects:

- If there is no infimum set already defined, then Infimum = ($\emptyset$, $\{b_1 \cup ... \ b_n\}$), where b_1 .. b_n are all attributes of context K.
- If there is no supremum set already defined, then Supremum = ($\{a_1 \cup ... \ a_n\}$, $\emptyset$), where a_1 .. a_n are all objects of context K.

4 RESULTS AND DISCUSSION

Before executing the In-Close 4 algorithm, a normalization of values was performed so that its execution was possible.

Normalization consists of removing the words *object* and *attribute* and placing the corresponding letter or index. For objects, we assign a set of numbers, and for attributes, a set of letters. Ex: $object[0] \rightarrow 1$ $attribute[5] \rightarrow F$

To perform the tests, 100 synthetic contexts were generated using *SCGaz*, varying the database with 10,000, 25,000, 50,000, and 100,000 objects, 50 attributes, and 50% of density.

Once the formal contexts were generated, they were subdivided into 2, 4 and 8 subgroups, to affirm that the entire generation of formal concepts is possible. Finally, five different formal contexts were generated for each dataset to guarantee full coverage of the tests.

Our tests were executed on a machine with an 8-core Mac M1 processor and 8 GB of main memory. The operating system used is a MacOS Mojave. In all tests performed, the formal concepts generated from the quasi-concepts, separating them into subsets, were equivalent to the original set of formal concepts.

The performance evaluation is based on the execution times of the serial version of In-Close 4 and our parallel proposal in C++/OpenMP.

Table 6 shows the results of a sub-division of the formal context (10,000x50) into subcontexts, i.e., the number of threads. The highest speedup[4] was 1.4246x for two threads, representing an efficiency[5] of the parallel version of 71.23% in comparison to the serial In-Close 4.

It is important to notice that when we increase the dataset size, this execution time increases due to the large number of incidences to be analyzed. Another point to consider is that there was a better result when subdivided into two groups (two threads).

Table 6: Results for 10,000 objects and 50 attributes.

# Concepts	In-Close 4	Time (s)	Parallel approach	# Threads	Time (s)
282,115		0.4781		2	0.3356
				4	0.3374
				8	0.359

Table 7 presents a formal concept of the same number of attributes, but the number of objects provided increased to 25,000. For this context, the highest speedup (two threads) was 1.4250x with an efficiency of 71.25%.

In Table 8, although the number of objects in-

[4]Speedup = serial time / parallel time

[5]Efficiency = speedup / number of cores

Table 7: Results for 25,000 objects and 50 attributes.

# Concepts		Time (s)		# Threads	Time (s)
298,512	In-Close 4	1.7656	Parallel approach	2	1.239
				4	1.251
				8	1.323

creased to 50,000, the highest speedup can be observed with two divisions (two threads), which was 1.4943x. The efficiency represents 74.71%.

Table 8: 50,000 objects and 50 attributes.

# Concepts		Time (s)		# Threads	Time (s)
428,215	In-Close 4	9.2264	Parallel approach	2	6.17445
				4	6.89105
				8	7.41605

In Table 9, the number of objects was increased to 100,000 with higher performance compared to serial In-Close 4. The highest speedup for this scenario (two threads) was 1.4116x with an efficiency of 70.48%.

Table 9: 100,000 objects and 50 attributes.

# Concepts		Time (s)		# Threads	Time (s)
582,219	In-Close 4	21.7264	Parallel approach	2	15.39105
				4	16.4178
				8	17.07075

In Figures 1 to 4, we see charts representing the performance as the number of divisions (threads) increased. The x-axis is the number of divisions (threads), and the y-axis is the execution time for both algorithms (serial In-Close 4 with 1 thread and our parallel version with 2, 4, and 8 threads). These figures show that we have scalability when the number of objects increases. However, more than two threads do not scale well for the same workload.

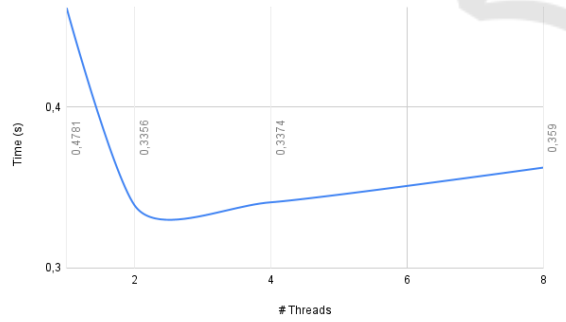

Figure 1: Execution time for 10,000 objects, 50 attributes and 0.5 of density.

5 RELATED WORK

Nilander (de Moraes et al., 2016) proposed a parallel implementation of the *NextClosure* algorithm for its optimization, resulting in a performance gain by reducing the execution time. Nilander's idea was

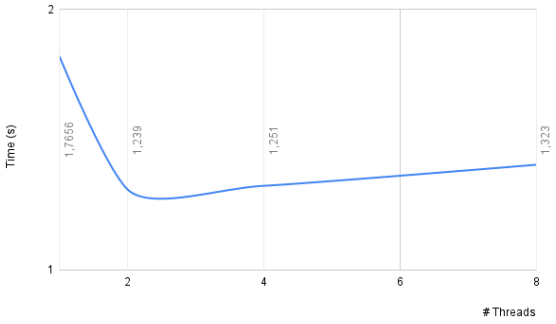

Figure 2: Execution time for 25,000 objects, 50 attributes and 0.5 of density.

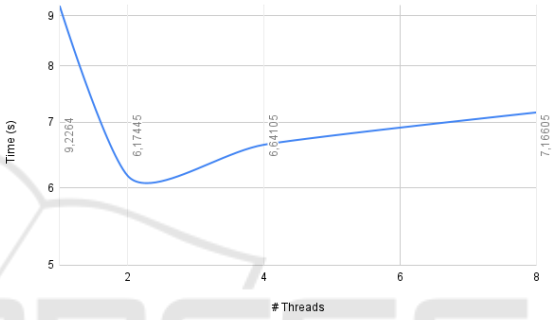

Figure 3: Execution time for 50,000 objects, 50 attributes and 0.5 of density.

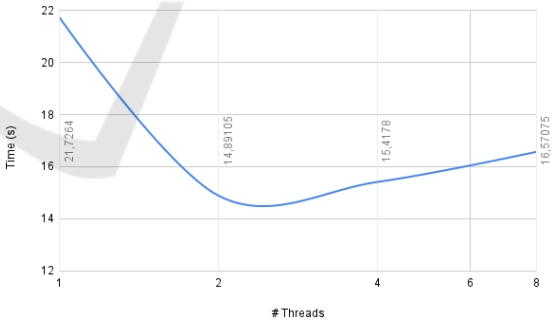

Figure 4: Execution time for 100,000 objects, 50 attributes and 0.5 of density.

proposed due to the large execution time demanded when executing the algorithm using high-dimensional databases. As a result, Nilander states that a considerable *speedup* gain was obtained from the algorithm but limited to the number of 16 concurrent *threads*.

Kodagoda *et al.* (Kodagoda et al., 2017) proposed a parallel version of the In-Close algorithm to reduce the running time of version 3 of the algorithm, using the *OpenMP API*. The authors claim that algo-

rithms based on CBO (*Colliding Bodies Optimization*) that are recursive by nature can easily be parallelized, which has become one of the pillars of making the project viable. A point to emphasize is using the *OpenMP*, which can be exploited to parallelize other algorithms. As a point of improvement, the authors propose that more comparisons be made with other parallel algorithms.

Novais *et al.* (Novais et al., 2021) proposed an algorithm for extracting formal concepts based on the *OpenCL* algorithm to obtain better performance for extracting them. The algorithm uses a parallel brute force approach, using a heterogeneous architecture (CPU+GPU and CPU+FPGA[6]). According to Novais *et al.*, the approach obtained a performance 18x better, in the best case, than the processing carried out by the *Data-Peeler* algorithm in the same context. They also highlighted that there were results in better energy efficiency since, in the best case, they obtained a gain of 1.79 operations performed by energy consumption compared to other algorithms in the different architectures the project covers.

Fu and Nguifo (Fu and Nguifo, 2004) also proposed an algorithm for obtaining concepts from search spaces in parallel. In such a proposal, the redundant attributes are removed from the formal input context before the partition generation step to reduce the algorithm's workload in extracting formal concepts. Regarding another important feature of the algorithm proposed by the author, it is based on the *NextClosure* algorithm to obtain the collection of concepts effectively. Furthermore, the author's approach does not consider balancing the workload, which is one of the limiting factors of the adopted strategy. In this work, the author concludes that parallel proposals can handle high-dimensional input contexts. But, due to unbalanced workloads, the algorithm may present sudden performance variations according to the selected load factor.

Zhou *et al.* (Zhou et al., 2021) proposed an improved version of In-Close 4, the so-called In-Close5. The new algorithm stores a concept's context and extension as a vertical bit matrix. Within the In-Close 4 algorithm, the context is stored just as a horizontal bit array, which is very slow in finding the intersection of two span sets. According to the authors, the experimental results show that the proposed algorithm is much more effective than the In-Close 4 algorithm, working even in contexts where In-Close 4 does not produce results.

In summary, Kodagoda *et al.* (Kodagoda et al., 2017) created a parallel version of In-Close 3. Zhou *et al.* (Zhou et al., 2021) work also modified the original algorithm, creating a new version called In-Close 5. Nilander, Novais *et al.*, Fu and Nguifo (de Moraes et al., 2016; Novais et al., 2021; Fu and Nguifo, 2004) used different algorithms and architectures, such as GPU and FPGA. Our work differs from the others since we developed an approach that uses a wrapper to run In-Close 4 algorithm.

[6]Field-Programmable Gate Array (FPGA)

6 THREATS TO VALIDITY AND LIMITATIONS

We discussed the potential for parallelization in generating formal concepts by dividing their context. The experimental results showed that parallelization could be achieved in concept generation.

However, this paper focuses on working with controlled synthetic datasets with the same number of attributes (50) and density (50%), but with different numbers of objects from 10,000 to 100,000. New results with a greater diversity of densities and attributes are already being analyzed.

We observed possible overheads in the execution of the proposed operations since several loops are made to verify objects and attributes. In addition, quasi-concepts must be joined, representing an additional computation. So, we are evaluating how to reduce overheads to increase performance and scalability.

7 CONCLUSIONS

This work successfully achieved its objective of parallelizing the generation of formal concepts by partitioning the formal context. We achieved this by developing a mathematical formalization and experimentation, improving speedup, e.g., of up to 1.4116x with an efficiency of 70.48% for 100,000 objects and 50 attributes with a density of 50%.

However, this work only addressed the division of formal contexts by groups of objects, requiring another analysis and adaptation to support the division by attributes.

For future work, we intend to execute more tests in environments with different architectures and configurations, such as measuring whether there was a speedup for different densities, attributes, and objects. We also consider evaluating energy consumption and the usage of heterogeneous architectures and other resources of parallelization.

ACKNOWLEDGEMENTS

The present work was carried out with the support of the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Financing Code 001 and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) - APQ-01929-22. The authors also thank CNPq and PUC Minas for their partial support in the execution of this work.

REFERENCES

Andrews, S. (2009). In-close, a fast algorithm for computing formal concepts. *International Conference on Conceptual Structures*.

Andrews, S. and Orphanides, C. (2010). Analysis of large data sets using formal concept lattices. *7th International Conference on Concept Lattices and Their Applications*, pages 104–115.

Carpineto, C. and Romano, G. (2004). *Concept data analysis: Theory and applications*. John Wiley & Sons.

Cerf, L., Besson, J., Robardet, C., and Boulicaut, J.-F. (2008). Data-peeler: Constraint-based closed pattern mining in n-ary relations. In *proceedings of the 2008 SIAM International conference on Data Mining*, pages 37–48. SIAM.

Chunduri, R. K. and Cherukuri, A. K. (2019). Scalable formal concept analysis algorithms for large datasets using spark. *Journal of Ambient Intelligence and Humanized Computing*, 10(11):4283–4303.

de Moraes, N. R., Dias, S. M., Freitas, H. C., and Zárate, L. E. (2016). Parallelization of the nextclosure algorithm for generating the minimum set of implication rules. *Artif. Intell. Research*, 5(2):40–54.

Domingos-Silva, J. P. and Vieira, N. J. (2008). A classification algorithm based on concept similarity. In *Research and Development in Intelligent Systems XXIV: Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 281–291. Springer.

Fu, H. and Nguifo, E. M. (2004). A parallel algorithm to generate formal concepts for large data. In *Concept Lattices: Second International Conference on Formal Concept Analysis, ICFCA 2004, Sydney, Australia, February 23-26, 2004. Proceedings 2*, pages 394–401. Springer.

Ganter, B., Stumme, G., and Wille, R. (2005). *Formal concept analysis: foundations and applications*, volume 3626. Springer.

Ganter, B. and Wille, R. (2012). *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.

Godinho, J., Gomide Gomes, J., Malheiro, R., and Santana, L. (2022). Hydrological forecast in macaé river basin with neural networks. *Revista Brasileira de Computação Aplicada*, 14(1):70–80.

Jindal, R., Seeja, K., and Jain, S. (2020). Construction of domain ontology utilizing formal concept analysis and social media analytics. *International Journal of Cognitive Computing in Engineering*, 1:62–69.

Kodagoda, N., Andrews, S., and Pulasinghe, K. (2017). A parallel version of the in-close algorithm. In *2017 6th National Conference on Technology and Management (NCTM)*, pages 1–5. IEEE.

Linden, R. (2008). *Algoritmos genéticos (2a ediçao)*. Brasport.

Mattson, T. G., He, Y., and Koniges, A. E. (2019). *The OpenMP Common Core Making OpenMP Simple Again*. The MIT Press.

Novais, J. P., Maciel, L. A., Souza, M. A., Song, M. A., and Freitas, H. C. (2021). An open computing language-based parallel brute force algorithm for formal concept analysis on heterogeneous architectures. *Concurrency and Computation: Practice and Experience*, page e6220.

Pacheco, P. (2011). *An introduction to parallel programming*. Elsevier.

Priss, U. (2006). Formal concept analysis in information science. *Annual review of information science and technology*, 40(1):521–543.

Quinn, M. J. (2003). Parallel programming. *TMH CSE*, 526:105.

Rimsa, A., Song, M. A., and Zárate, L. E. (2013). Scgaz-a synthetic formal context generator with density control for test and evaluation of fca algorithms. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3464–3470. IEEE.

Robey, R. and Zamora, Y. (2021). *Parallel and high performance computing*. Simon and Schuster.

Van Der Pas, R., Stotzer, E., and Terboven, C. (2017). *Using OpenMP-The Next Step: Affinity, Accelerators, Tasking, and SIMD*. The MIT Press.

Zhou, J., Yang, S., Wang, X., and Liu, W. (2021). A new algorithm based on extent bit-array for computing formal concepts. *arXiv preprint arXiv:2111.00003*.