Generic software for benchmarking Formal Concept Analysis: Orange3 integration

Nicolas Leutwyler[1,2,3], Mario Lezoche[1,2], Hervé Panetto[1,2,] and Diego Torres[3,4]

[1] Université de Lorraine, Nancy 54000, France
[2] CRAN, Nancy 54000, France
[3] LIFIA, CICPBA-Facultad de Informática, UNLP, La Plata 1900, Argentina
[4] Dto. CyT, UNQ, Bernal, Argentina

Abstract. Thanks to the internet of things (IoT) and cyber physical systems (CPS), we face an incremental growth of the available data, either on the internet or in private databases. This resulted in data mining techniques becoming an essential piece in the information retrieval process. Moreover, trends like the industry 4.0 encourages its usage to support data driven decisions, for instance. Formal Concept Analysis (FCA) is one of the most used techniques in the unsupervised data mining field due to its inherent ability to find patterns between concepts. As a consequence, many applications need the use of fast algorithms to perform the calculations to retrieve either the lattice or the association rules related with the data at their disposal. Due to this, scientists often rely on manually crafted benchmarks to compare how certain algorithms perform under different circumstances. In this work, we propose the architecture of a software to generalize these benchmarks independently of the algorithms, to be integrated in the open source data analysis software Orange3.

Keywords: Formal Concept Analysis · benchmarking · metaprogramming · open source.

1 Introduction

Data mining techniques are widely used to support data driven decisions [19], to infer knowledge *automatically* in contexts such as software modelling, or artificial intelligence [25,24]. Furthermore, the internet of things and cyber physical systems could use some gains the data mining provide through semantic interoperability [3,20]. However, these fields need precise and fast processing of the information since they usually work in real time [22]. In this context, the effort towards finding fast data mining algorithms and measuring their performance is understandable [13,14].

In conventional agriculture, pesticides, antimicrobials and other pest control products are undesired, thus the need of alternative solutions. The European Green Deal is one of the most important actions in Europe to overcome the challenges of climate change and environmental degradation, sometimes caused by the usage of pesticides. In the state of the art, there are numerous descriptions of active plant-based products used as bio-pesticides. The Knomana (KNOwledge MANAgement on pesticide plants in Africa) project's goal is to gather data

Received April 2023 Accepted July 2023; Published August 2023

about these bio-pesticides and implement methods to support the exploration of knowledge by the potential users (farmers, researchers, retailers, etc.). Considering the needs expressed by the domain experts, information retrieval is needed to obtain relevant insight on the matter. In addition, data clustering into similar groups is helpful when it comes to understanding key differences (or similarities) of objects in general. Formal Concept Analysis (FCA) appears as a suitable approach, due to its inherent qualities for structuring and classifying data through conceptual structures that provide a relevant support for data exploration.

Additionally, the trend driven by the Industry 4.0 [4,15] is to increase the usage of the available data in order to increase the performance and the efficiency of processes. In particular, regarding the Agriculture 4.0, impulsed also by the Green Deal in the EU, several works have been carried out using the data mining method called Formal Concept Analysis (FCA) [23,9], and also its multi-relational data mining [7] extension Relational Concept Analysis (RCA) [21,11]. There are several good results about the time complexity in the worst case from the main FCA algorithm, which is the one that calculates the set of *formal concepts* (see Section 3). Nevertheless, according to some previous experiments, it is known that some algorithms with worse time complexity than others perform better under certain circumstances [13,14].

Consequently, while developing new algorithms in this area, it is important to also perform a good benchmarking suit of tests to understand in which situations the algorithms strives. This is something that usually takes extra effort since it is for the most part a manual process. In the current literature, one way to ease the manual work required to perform these tests is approached by providing generic testing tools for the particular application [17,26]. Particularly, we could not find any work in this area applied particularly to the algorithms for FCA and its extensions.

In this work, we introduce a software tool to benchmark, and another to use FCA in data pipeline. Both of them are thought to be added to the architecture of the open access data analysis software Orange3 [18]. It is worth mentioning that some works have been published in the field of generically benchmarking algorithms [6,5], and that our goal in this paper is to present a tool that, while generic, it still provides specific functions for the FCA use case.

The paper is organized as follows: In Section 2, we discuss the state of the art of generic testing tools for specific applications. Section 3 explains the notation and concepts we will use throughout the document. Section 4 presents a use case of the Formal Concept Analysis as a motivation for the creation of the generic tool. Section 5 presents the software model, the context in which it is integrated, and the main algorithms. Finally, Section 6 summarizes the contribution and discusses the possible future work.

2 Related Work

For the purpose of this work, we consider that testing tools are divided in two categories. The first one, is a one in which the tool must provide a set of options

Received April 2023 Accepted July 2023; Published August 2023

to reliably test a specific process that never changes. For example, a tool to automatically test REST API's load. Regardless of the specific endpoints, the testing part would always follow the pattern of reaching the endpoints, measuring the time between request and response, and so on and so forth [8,28]. The second group is the one that involves giving to the user a generic set of functions to test something that we do not know in advance, e.g., testing algorithms, functions in general. The challenge of this category is the fact that the process we want to test is not known beforehand, and thus, the techniques used to solve them usually involve metaprogramming or reflection [16]. A commonly applied method to tackle this type of tools, is to develop a *domain specific language* (DSL) in order to provide the users a flexible way to define what or how to test their functions [1,12].

In the case of benchmarking the calculation of formal concepts in FCA, the problem belongs to both categories. On the one hand, the main process will include one step that will always be a part of it: calculating derivatives (explained in Section 3). On the other hand, how or when the algorithm will do it is unknown and hence it belongs to the second category. Therefore, the solution we propose includes a part that explodes the common pattern the method will always follow, and one in which the user is given the possibility to manually choose what and how to test. The disadvantage of the solution, is that it will not completely remove the manual effort required, but since the complexity is encapsulated in the provided functions, it will reduce it.

3 Preliminaries

3.1 Formal Concept Analysis

Formal Concept Analysis (FCA) is a clustering method whose input is a triple $K = (O, A, I)$, where O is a set of objects, A is a set of attributes, and I is an incidence matrix, indicating whether each object has an attribute or not e.g., $K = (\{o_1, o_2\}, \{a_1, a_2\}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix})$ is a formal context in which o_1 only has the attribute a_2, and o_2 has both. Alternatively we can see it as a *bipartite graph* i.e., O and A are the disjoint sets of *nodes*, and I is the set of *arcs*. The derivative operation $'$ on objects in the set $X \subseteq O$ is defined as the intersection of attributes of each object $o \in X$.

$$X' = \{a \in A \mid \forall o \in X : I_{o,a}\} \tag{1}$$

Analogously, we can define the derivative of a set of attributes as follows,

$$Y' = \{o \in O \mid \forall a \in Y : I_{o,a}\} \tag{2}$$

Having this in mind, a *formal concept* is a pair $C = (X, Y)$ where $X \subseteq O$, $Y \subseteq A$ such that $X' = Y$, $Y' = X$. X is called the extent and Y the intent. Put it into bipartite graph notation, a formal concept is a *bi-clique* i.e., a complete bipartite subgraph. For readability purposes, we note $C.E$ to the extent, and $C.I$ to the

Received April 2023 Accepted July 2023; Published August 2023

intent. The set of all the formal concepts and the relation of inclusion of extents form the so-called *concept lattice*, which is a partially ordered set, and is often noted with the letter $\mathcal{L}$.

3.2 Common algorithms and their differences

Many reviews about algorithms for computing formal concepts have been made in the past [2,14,27]. There are many nuances to how they are implemented and also to their output. Some of them compute only the formal concepts, whilst others also calculate their underlying lattice diagram. For the purpose of this work, we will consider computing all the formal concepts and calculating their diagram to be separate problems, although they can be solved at the same time.

As we mentioned in Section 1, and as we can see in the mentioned reviews, there are many approaches on how to deal with the repetition of results in the calculation of the concepts, which occurs mainly because different subsets $X \subseteq O$, might yield the same $Y \subseteq A$ when the derivative operation is applied, but only the largest of them is present in a concept. Algorithms deal with this problem in different ways, from which in this work we aim to consider two: having a clever structure that allows to rapidly finding repeated results (e.g., Linding's algorithm, etc), or by traversing the context in a certain order that ensures that some results will not be repeated (e.g., Andrews' Inclose algorithm, etc).

3.3 Orange3 software

The *Orange3* software [18] is an open source machine learning and data visualisation tool whose aim is to make data analysis accessible to the end user in an intuitive way. To achieve this, it provides a way to pipeline data through "boxes" with certain input and output each, allowing to reuse them whenever necessary. This structure also allows simplifying the way to contribute to the project, since boxes can be thought as independent programs that define how to interact with their input and how to export the output. Additionally, Orange3 allows the development of separate plugins or *add-ons*, or in other words: external pieces of software that can be added to the main application. The scope of this paper is to introduce the architecture of an add-on with its components and interactions, and explaining how it would help to the analysis and benchmarking of formal concept analysis algorithms.

4 Motivation

To have a closer idea to what the Knomana dataset contains, an extract can be found in the Table 1. The formal context's objects are names of organisms composed by three parts: species, genus, and family. Even though they could be considered as different types, i.e., crops, pests, and protection species, in this example, we put them in the same table because they share the same attributes.

Received April 2023 Accepted July 2023; Published August 2023

Table 1: Plants, crops and bio-aggressors formal context

K	**Food**	**Medical**
Abies sibirica/ Abies/ Pinaceae		
Acanthospermum hispidum/ Acanthospermum/ Asteraceae		X
Anticarsia gemmatalis/ Anticarsia/ Noctuidae		
Allium sativum/ Allium/ Amaryllidaceae	X	X
Spodoptera frugiperda/ Spodoptera/ Noctuidae		
Spodoptera littoralis/ Spodoptera/ Noctuidae		
Spodoptera litura/ Spodoptera/ Noctuidae		
CropS/ CropG/ CropF	X	X
CropFabaS/ CropFabaG/ Fabaceae	X	
Zanthoxylum rhetsa/ Zanthoxylum/ Rutaceae		X
Zingiber officinale/ Zingiber/ Zingiberaceae	X	X

Particularly, in the work [10], the FCA extension RCA is used to extract patterns in the data related to some plants being natural pesticides to other ones. Moreover, RCA needs to perform the algorithm to calculate the set of formal concepts of formal contexts many times in a loop until it converges, as explained in the Figure 1. Each iteration, using the calculated lattices and the relations in the input, increases the size of formal contexts in terms of their attributes, i.e., adds columns. This results on the possibility of the number of formal concepts increasing greatly, hence the need of fast algorithms to do it. In addition, available algorithms to calculate formal concepts perform differently according to the type of data, for instance, some of them perform better when formal contexts are sparse, and others when they are dense.

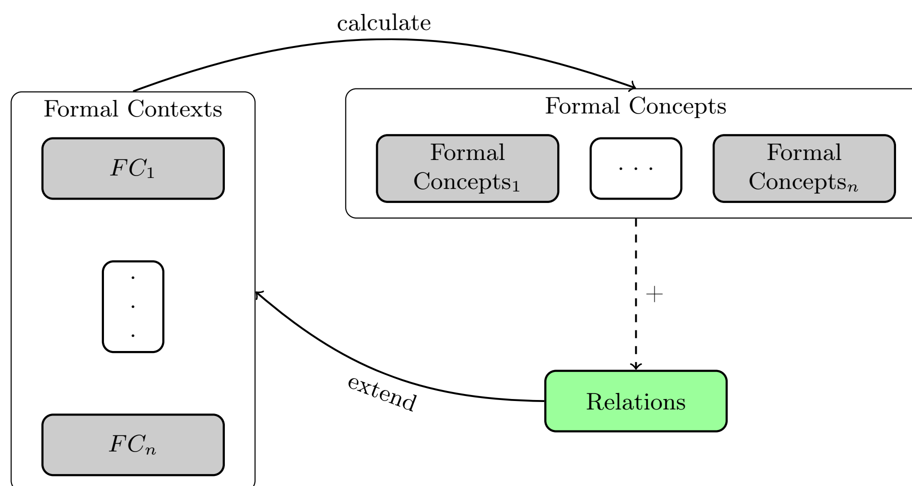

Fig. 1: RCA extension algorithm main loop

Received April 2023 Accepted July 2023; Published August 2023

This leads us to a state in which making all kinds of experiments on these algorithms is necessary to better understand when which of them are more suitable according to the situation. However, it is rather tedious, since algorithms are typically different between themselves and there is not a common pattern to test them all the same way. Therefore, this work aims to smoothen the effort needed to perform benchmarking experiments on particularly formal concepts calculation algorithms. Additionally, the paper is intended to serve as a first approach guide on how to tackle generic denotational testing frameworks using metaprogramming techniques.

5 Software model

As explained in the Section 3.3, the units that users have to deal with in the platform are represented by *boxes* that are in fact algorithms with defined *inputs* and *outputs*. In that regard, the first input that concerns us is the name of a file representing a Formal Context, which, in our particular case, will be a *csv*. For the mentioned input, there should be a box called Formal Context that outputs the parsed formal context $K = (O, A, I)$ so that other algorithms do not have to deal with the parsing task over and over again. Then, to provide visualization to what we are parsing, there will be a box whose purpose is to show the bipartite graph representation of K (see Section 3). Furthermore, and continuing with the visualization, the add-on will provide a box for visualizing the Hasse diagram (and leave the door open to implement other visualizations such as the Iceberg concepts lattice).

Regarding the core and therefore the most important part of the architecture, the plugin will include a box that computes the list (or stream) of formal concepts, and that will act as the entry point for the generic benchmarking abstract public interface (a.k.a., API). This box (red one in the Fig 2) will allow executing a default algorithm (Inclose), or to choose a user defined one. There is where, depending on what are the metrics the user specified to measure, the box will show them in a table format. The goal is to provide the user a set of generic decorators that allow to annotate specific functions, or specific parts of the algorithm to be measured in different ways: how many times a certain function is executed, how much memory it consumes, how much time in total or in average it spent during the execution, etc. And some specific ones related to formal context analysis, such as the times a derivative was calculated repeatedly.

5.1 Benchmarking API

The benchmarking API will heavily rely on introspection and meta-programming patterns [16]. Particularly, it will use decorators to add meta information to the algorithm that will later be used for the *runner* to gather the data and be able to output it in some fashion. The notation we will use for the decorators is the same *Python* uses, and it consists of adding an @ to the beginning of each of them in order to identify them. Additionally, in the pseudocode, we will use

Received April 2023 Accepted July 2023; Published August 2023

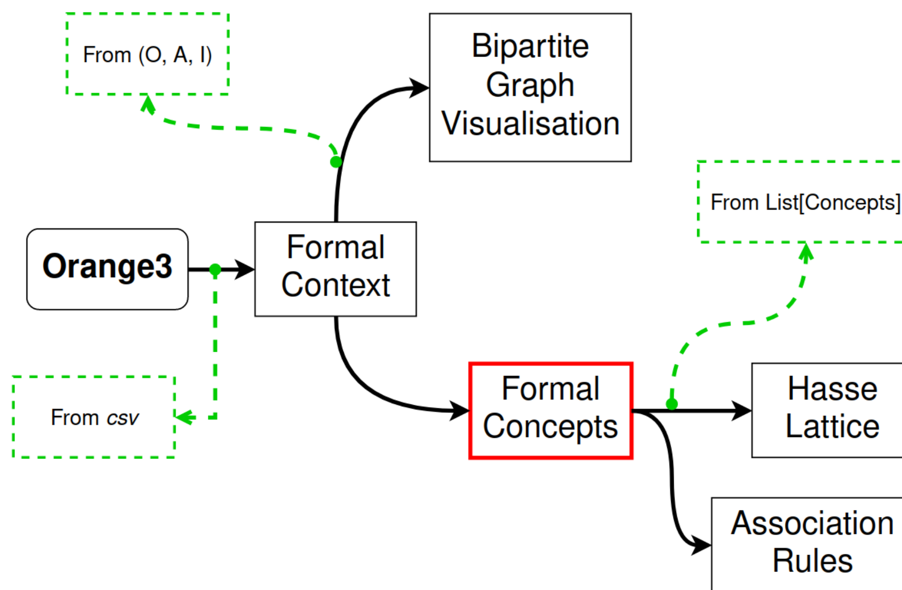

Fig. 2: Architecture diagram representing the expected components and their interactions

*args as a way to say "any number of parameters". The generic decorators will be the following,

1. *@measure_time()*
2. *@measure_times_executed()*
3. *@measure_memory()*

while the FCA specific ones will be,

4. *@object_derivative()*
5. *@attribute_derivative()*

Specifically, the decorator 1 is expected to be applied to any function the user would want to measure the time it takes. In addition, the decorator 2 will count how many times a function is executed. And finally, the 3rd decorator will take note of the memory usage during the execution of the function and output the maximum usage of it. Notice that the *runner* will measure every decorated function, even if it is being called recursively, meaning that depending on what the user wants to do, sometimes it would be better to separate recursive functions in the *first call* and then the recursive one. Decorators 4 and 5, both describe a function that, each time is called, produces an object or an attribute derivative respectively. This provides essential information to the *runner* to measure how many times the overall algorithms repeats calculations.

Received April 2023 Accepted July 2023; Published August 2023

Implementation Firstly, we will use the words *decorator* and *wrapper* interchangeably. On the one hand, the three first decorators are very similar in structure: they will wrap the function with a specific type of function or class that hints the runner to call it in a particular way and also output their specific type of information, e.g., a float representing time, an integer representing the amount of times the function has been executed, etc. When the wrapper is called, it initializes the necessary objects to gather the information, then it calls and returns the same as the function it wraps. As the execution ends, the wrapper will have the information saved in a dictionary as an instance variable to be collected by the *runner*.

On the other hand, the fourth and fifth decorators are different in the sense that they have to also keep track of what are the structures already generated. To do that, it is necessary to implement a way to tell whether two structures are the same or not.

Algorithm 1: *measure_time* decorator

Input: f, a function or a *callable* class
Output: A callable class responsible for measuring the execution time of f

```
1 Def wrapper(*args, collector):
2 |   start_timer()
3 |   res ← f(*args)
4 |   time_passed ← stop_timer()
5 |   collector.add_measure_time(f, time_passed)
6 |   return res
7 end
8 return wrapper
```

Algorithm 2: *measure_times_executed* decorator

Input: f, a function or a *callable* class
Output: A callable class responsible for measuring how many times f is called

```
1 Def wrapper(*args, collector):
2 |   res ← f(*args)
3 |   collector.add_times_executed(f)
4 |   return res
5 end
6 return wrapper
```

Received April 2023 Accepted July 2023; Published August 2023

Algorithm 3: *measure_ memory* decorator

```
  Input: f, a function or a callable class
  Output: A callable class responsible for measuring the maximum memory f
          consumes during its executions
1 Def wrapper(*args, collector):
2 |  profiled_function ← profile(f)
3 |  res, profiling_ data ← profiled_function(*args)
4 |  collector.function_ executed_ with_ memory(f, profiling_ data.memory)
5 |  return res
6 end
7 return wrapper
```

Algorithm 4: *object_ derivative* decorator

```
  Input: f, a function or a callable class
  Output: A callable class responsible for measuring the amount of times a set
          has been calculated
1 Def wrapper(*args, collector):
2 |  res ← f(*args)
3 |  collector.object_ derivative_ calculated(res)
4 |  return res
5 end
6 return wrapper
```

5.2 Runner

This component is the responsible for running the user provided code as its name suggests, but also for reporting the information at the end of the run. Thanks to the fact that decorators handle the complexity of knowing *when* the functions are executed, and also *what* to do in each case, for the running part, the runner should only execute the algorithm as it is. The challenge comes in the collecting part because the runner does not have control over when *some* of the functions are called, in fact, most of them will be instantiated to be executed and then discarded and the runner would not even notice it. To solve this problem, all decorators will expect one more parameter besides the function to wrap, being an object whose purpose is to save each measurable function call. Then, our runner will run a modified version of the abstract syntax tree (AST) that provides this parameter, having access to this new object, and thus having access to the information after its execution.

Particularly, the Algorithm 1 starts a timer, runs the *original function* with its parameters, after, it measures the time that passed between the call and the end of the function, and finally, it tells the *collector* to add it to the total time spent for that specific function. Following the same pattern, the Algorithm 2, simply executes the function with its parameters, and afterwards it adds one to the total times executed for the specific function. And somehow more complex, the Algorithm 3 wraps the function to be executed to a profiling wrapper,

Received April 2023 Accepted July 2023; Published August 2023

and after executing it, sends the memory statistic to the *collector*. Lastly, the Algorithm 4 calls the collector's *object_derivative_calculated* function, to add 1 to the amount of times that particular derivative has been calculated. The algorithm for *attribute_derivative* would be exactly the same as 4 but calling the function that adds to the attribute derivatives instead. It is important to notice that all these algorithms return a function defined inside it, meaning it is considering high order functions, i.e., functions as first class citizen values.

On top of each wrapper, the collector is expected to be an object with the following methods

Algorithm 5: *add_measure_time collector*'s method

Input: f, a function or a *callable* class, and t, an integer representing the time spent by f

Output: A mehod responsible for adding the time spent by f

1 times_table[f] += t

Algorithm 6: *add_times_executed collector*'s method

Input: f, a function or a *callable* class

Output: A mehod responsible for adding 1 to f times executes

1 times_executed_table[f] += 1

Algorithm 7: *function_executed_with_memory collector*'s method

Input: f, a function or a *callable* class, and m an integer representing the memory spent by f

Output: A mehod responsible for recording the most memory spent by the executions of f

1 memory_spent_table[f] $\leftarrow$ max(memory_spent_table[f], m)

All three algorithms assume the existence of a mapping between functions and their specific value. In particular, the Algorithm 5 adds t to the mapping, here the algorithm assumes that the table has been previously initialized with 0, resulting in a semantic that will maintain the total amount of time spent by a function. In the same line, the Algorithm 6 adds 1 to the current value, meaning that it correctly counts how many times a function has been executed. Finally, the Algorithm 7 always remembers the maximum between what it had previously and the new m.

Received April 2023 Accepted July 2023; Published August 2023

6 Conclusion and future work

In this work, we presented a generic tool to allow benchmarking certain aspects of FCA formal concepts generation algorithms instead of handcrafting them each time. The tool is currently being developed on their PIDIR, by Soukayna Ouabi and Loïc Chaillot, two students at TELECOM Nancy Engineering School. The advantages of the tool are not only the encapsulation and the centralization of the benchmarking complexities, but also the fact that it provides the programmers a denotational way to mark the parts of the code they want to benchmark, i.e., they write *what* instead of *how*.

Furthermore, the PIDIR project is expected to be extended in the coming months, so the tool can be upgraded to also include FCA extensions such as the widely used Relational Concepts Analysis (RCA). This will come with its own challenges, mainly in the area of software modelling. On top of that, many challenges are still open in the development of a tool whose main goal is to be versatile and to provide an easy data visualization to the user. Combining that, plus the fact that both FCA and RCA produce an output with an exponential size in terms of their input, we realize that much work could be done in order to guarantee that all data can be explored, without the need of having everything loaded in memory.

Finally, the work as a whole could be considered as an approach to tackle problems involving the generation of tools for programmers and scientists working on algorithms creation and benchmarking in general, since the form to proceed should be in the lines of: understanding what needs to be tested, generating the denotational API, adding the collector, and when the time to benchmark is needed, the runner should always run a modified version of the AST adding the extra parameter to all the places needed, e.g., all functions decorated with the API decorators.

References

1. Albonico, M., Benelallam, A., Mottu, J.M., Sunyé, G.: A DSL-based approach for elasticity testing of cloud systems. In: Proceedings of the International Workshop on Domain-Specific Modeling. pp. 8–14. DSM 2016, Association for Computing Machinery, New York, NY, USA (Oct 2016). `https://doi.org/10.1145/3023147.3023149`, `https://doi.org/10.1145/3023147.3023149`
2. Arévalo, G., Berry, A., Huchard, M., Perrot, G., Sigayret, A.: Performances of Galois Sub-hierarchy-building Algorithms. In: Kuznetsov, S.O., Schmidt, S. (eds.) Formal Concept Analysis. pp. 166–180. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2007). `https://doi.org/10.1007/978-3-540-70901-5_11`
3. Camarinha-Matos, L.M.: Challenges in IoT Applications and Research. Internet of Things. Technology and Applications. IFIPIoT 2021 (Jan 2022), `https://www.academia.edu/73315599/Challenges_in_IoT_Applications_and_Research`
4. Cardin, O.: Contribution à la conception, l'évaluation et l'implémentation de systèmes de production cyber-physiques. Habilitation à diriger des recherches, Université de nantes (Dec 2016), `https://tel.archives-ouvertes.fr/tel-01443318`

Received April 2023 Accepted July 2023; Published August 2023

5. Darmont, J., Schneider, M.: Benchmarking OODBs with a Generic Tool. J. Database Manag. **11**, 16–27 (Jul 2000). https://doi.org/10.4018/jdm.2000070102
6. Digalakis, J., Margaritis, K.G.: On benchmarking functions for genetic algorithm. International Journal of Computer Mathematics **77** (Jan 2001). https://doi.org/10.1080/00207160108805080
7. Džeroski, S.: Multi-relational data mining: an introduction. ACM SIGKDD Explorations Newsletter **5**(1), 1–16 (Jul 2003). https://doi.org/10.1145/959242.959245, https://doi.org/10.1145/959242.959245
8. Fertig, T., Braun, P.: Model-driven Testing of RESTful APIs. In: Proceedings of the 24th International Conference on World Wide Web. pp. 1497–1502. WWW '15 Companion, Association for Computing Machinery, New York, NY, USA (May 2015). https://doi.org/10.1145/2740908.2743045, https://doi.org/10.1145/2740908.2743045
9. Ganter, B., Stumme, G., Wille, R.: Formal Concept Analysis: Foundations and Applications. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2005), https://books.google.fr/books?id=oyb6BwAAQBAJ
10. Keip, P., Gutierrez, A., Huchard, M., Le Ber, F., Sarter, S., Silvie, P., Martin, P.: Effects of Input Data Formalisation in Relational Concept Analysis for a Data Model with a Ternary Relation. In: Cristea, D., Ber, F.L., Sertkaya, B. (eds.) ICFCA 2019 - 15th International Conference on Formal Concept Analysis. Lecture Notes in Computer Science, vol. 11511, pp. 191–207. Springer International Publishing, Frankfurt, Germany (Jun 2019). https://doi.org/10.1007/978-3-030-21462-3_13, https://hal-lirmm.ccsd.cnrs.fr/lirmm-02092148
11. Keip, P., Gutierrez, A., Huchard, M., Le Ber, F., Sarter, S., Silvie, P., Martin, P.: Effects of Input Data Formalisation in Relational Concept Analysis for a Data Model with a Ternary Relation. In: Cristea, D., Le Ber, F., Sertkaya, B. (eds.) Formal Concept Analysis. pp. 191–207. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-21462-3_13
12. King, T.M., Nunez, G., Santiago, D., Cando, A., Mack, C.: Legend: an agile DSL toolset for web acceptance testing. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. pp. 409–412. ISSTA 2014, Association for Computing Machinery, New York, NY, USA (Jul 2014). https://doi.org/10.1145/2610384.2628048, https://doi.org/10.1145/2610384.2628048
13. Kuznetsov, S., Obiedkov, S.: Algorithms for the Construction of Concept Lattices and Their Diagram Graphs, vol. 2168 (Sep 2001). https://doi.org/10.1007/3-540-44794-6_24, journal Abbreviation: Principles of Data Mining and Knowledge Discovery - Lecture Notes in Computer Science Pages: 300 Publication Title: Principles of Data Mining and Knowledge Discovery - Lecture Notes in Computer Science
14. Kuznetsov, S.O., Obiedkov, S.A.: Comparing performance of algorithms for generating concept lattices. Journal of Experimental & Theoretical Artificial Intelligence **14**(2-3), 189–216 (Apr 2002). https://doi.org/10.1080/09528130210164170, http://www.tandfonline.com/doi/abs/10.1080/09528130210164170
15. Lezoche, M., Hernandez, J.E., Alemany Díaz, M.d.M.E., Panetto, H., Kacprzyk, J.: Agri-food 4.0: A survey of the supply chains and technologies for the future agriculture. Computers in Industry **117**, 103187 (May 2020). https://doi.org/10.1016/j.compind.2020.103187, https://linkinghub.elsevier.com/retrieve/pii/S0166361519307584

Received April 2023 Accepted July 2023; Published August 2023

16. Lilis, Y., Savidis, A.: A Survey of Metaprogramming Languages. ACM Comput. Surv. (2020). https://doi.org/10.1145/3354584
17. Liu, Y., Liu, Y., Chen, T.Y., Zhou, Z.Q.: A Testing Tool for Machine Learning Applications. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp. 386–387. Association for Computing Machinery, New York, NY, USA (Jun 2020), https://doi.org/10.1145/3387940.3392694
18. Ljubljana, University of, B.L.: Orange3, https://orangedatamining.com/
19. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval (Jul 2008). https://doi.org/10.1017/CBO9780511809071, https://www.cambridge.org/highereducation/books/introduction-to-information-retrieval/669D108D20F556C5C30957D63B5AB65C, iSBN: 9780511809071 Publisher: Cambridge University Press
20. Panetto, H., Lezoche, M., Hernandez Hormazabal, J.E., del Mar Eva Alemany Diaz, M., Kacprzyk, J.: Special issue on Agri-Food 4.0 and digitalization in agriculture supply chains - New directions, challenges and applications. Computers in Industry **116**, 103188 (Apr 2020). https://doi.org/10.1016/j.compind.2020.103188, https://linkinghub.elsevier.com/retrieve/pii/S0166361519311145
21. Rouane-Hacene, M., Huchard, M., Napoli, A., Valtchev, P.: Relational Concept Analysis: Mining Concept Lattices From Multi-Relational Data. Annals of Mathematics and Artificial Intelligence **67** (Jan 2013). https://doi.org/10.1007/s10472-012-9329-3
22. Sayad, S.: Real Time Data Mining (Jan 2017)
23. Tamrakar, E.S.: Formal concept analysis: mathematical foundations (Jan 1997), https://www.academia.edu/3362029/Formal_concept_analysis_mathematical_foundations
24. Wajnberg, M.: Analyse relationnelle de concepts : une méthode polyvalente pour l'extraction de connaissance. Theses, Université du Québec à Montréal ; Université de Lorraine (Nov 2020), https://hal.archives-ouvertes.fr/tel-03042085, issue: 2020LORR0136
25. Wajnberg, M., Valtchev, P., Lezoche, M., Panetto, H., Massé, A.: Mining Process Factor Causality Links with Multi-relational Associations. p. 266. Marina Del Rey, CA, United States (Sep 2019). https://doi.org/10.1145/3360901.3364446
26. Yang, C.S.D., Pollock, L.L.: Towards a structural load testing tool. In: Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis. pp. 201–208. ISSTA '96, Association for Computing Machinery, New York, NY, USA (May 1996). https://doi.org/10.1145/229000.226318, https://doi.org/10.1145/229000.226318
27. Zaki, M., Hsiao, C.J.: Efficient algorithms for mining closed itemsets and their lattice structure. Knowledge and Data Engineering, IEEE Transactions on **17**, 462–478 (May 2005). https://doi.org/10.1109/TKDE.2005.60
28. Zhang, Y., Fu, W., Nie, C.: monadWS: a monad-based testing tool for web services. In: Proceedings of the 6th International Workshop on Automation of Software Test. pp. 111–112. AST '11, Association for Computing Machinery, New York, NY, USA (May 2011). https://doi.org/10.1145/1982595.1982622, https://doi.org/10.1145/1982595.1982622

Received April 2023 Accepted July 2023; Published August 2023