Scalable Performance of FCbO Update Algorithm on Museum Data

Tim Wray¹, Jan Outrata²*, and Peter Eklund¹

¹ IT University of Copenhagen
² Dept. Computer Science, Palacký University Olomouc, Czech Republic

Abstract. Formal Concept Analysis – known as a technique for data analysis and visualisation – can also be applied as a means of creating interaction approaches that allow for knowledge discovery within collections of content. These interaction approaches rely on performant algorithms that can generate conceptual neighbourhoods based on a single formal concept, or incrementally compute and update a set of formal concepts given changes to a formal context. Using case studies based on content from museum collections, this paper describes the scalability limitations of existing interaction approaches and presents an implementation and evaluation of the FCbO update algorithm as a means of updating formal concepts from large and dynamically changing museum datasets.

1 Introduction

Formal Concept Analysis is best known as a technique for data analysis, knowledge representation and visualisation. A number of case studies have been developed that also use FCA as a means of creating and visualising the semantic spaces within museum collections – allowing users to visualise, explore and discover new objects within these collections based on their associations and commonalities with other objects. Some of these applications include Virtual Museum of the Pacific [1], the Brooklyn Museum Canvas [2] and the A Place for Art [3] iPad app. These case studies led to the development of a set of web services called the COLLECTIONWeb framework [4, 5]. Their analysis gave rise to new interactions approaches based on FCA that required the use of fast algorithms for computing the upper and lower neighbours of a formal concept, and for computing and updating a set of formal concepts based on incremental changes to their formal contexts. These approaches are described as the conceptual neighbourhood approach and concept layer approach, respectively. This paper focuses on the implementation and scalability limitations of the conceptual neighbourhood approach, along with the FCbO update algorithm, its implementation within the concept layer approach and its performance evaluation.

* J. Outrata acknowledges support by the grant No. IGA_PrF_2016_027 of the IGA Palacký University Olomouc.

The case studies are motivated by emerging museological movements that have occurred since the 1970s that recognise the museum’s role in collecting, creating and shaping knowledge in which the context of an object has become an increasingly important part of its analysis, interpretation and communication. [6–9]. Context can refer to an object’s materials, construction, design, ornamentation, provenance, history, environment, connection to people and human society [9, 10]. This focus towards context reflects a shift from a classical worldview, where objects were classed in terms of order, hierarchy and taxonomy, to a modern perspective where objects are analysed in terms of links to other objects, people, social and cultural histories [9]. The natural association between these modern perspectives of information and knowledge sharing within museums are in accord with the foundations of Formal Concept Analysis in its ability to augment human thought, communication and interpretation. [11, 12]. This association motivates the research into new design and interaction approaches that emphasise concept generation and discovery within museum collections that rely on fast and efficient algorithms for computing formal concepts and their conceptual neighbours.

2 FCA algorithms: scalability and performance evaluation

2.1 The conceptual neighbourhood approach

In the museum-based case studies reported, FCA is used to provide conceptual structures that can be navigated by a user. The conceptual neighbourhood approach, as reported in [13], offers the ability to view individual concepts and move between neighbouring concepts within a concept lattice. One implementation of this approach is to compute and store a complete concept lattice that can then be traversed by the user. However as is well known, complete concept lattices – while adequate for visualising small datasets – are computationally prohibitive and visually complex on medium to larger datasets typically associated with museum collections that typically contain tens of thousands of objects [14].

The time and space complexities of pre-computing and storing a complete concept lattice can be understood by a discussion of how the approach scales with respect to the size of a formal context. Following an analysis of algorithms that build complete concept lattices, Carpineto and Romano [14] identify their time complexities: the best result being the ConceptSCover algorithm which has a worst-case time complexity of $O(|C||M|(|G| + |M|))$ which is dependent, in part, on the number of formal concepts generated from a formal context. The number of formal concepts $|C|$ generated from a formal context $K := (G, M, I)$, can be linear (in the best case) or quadratic (in the worst case) with respect to $|G|$ (the number of objects) or $|M|$ (the number of attributes) within the formal context depending on the number of attributes per object. However, even withstand the time and space complexities for initially computing and storing concept lattices from a large formal context (which, if the system employed update algorithms to update the concept lattice, would only need to be run once), the worst-case time complexity for updating a pre-computed concept lattice –
i.e., only computing a portion of a concept lattice given changes to a formal context – is quadratic with respect to the number of formal concepts \(|C|\); although experimental results \([15, 16]\) (cited in \([14]\)) suggest that in practice, the growth may be linear, rather than quadratic. Despite this, updating and storing a complete concept lattice for conceptual navigation poses major scalability and space concerns for large formal contexts.

**CollectionWeb** implements an alternate approach that does not require computation of the complete concept lattice and therefore negates the above scalability issues, but still allows the user to navigate between neighbouring formal concepts – via the reduction and inclusion of query attributes. This method, called the *conceptual neighbourhood approach*, was used in *ImageSleuth* \([13, 12]\) and again in the *Virtual Museum of the Pacific* \([1]\). In both cases interaction follows a partial view of the concept lattices in the form of a single formal concept and its immediate neighbours.

The algorithm used by **CollectionWeb** for generating conceptual neighbourhoods is the **NearestNeighbours** algorithm \([14]\), presented in Algorithm 1. The conceptual neighbourhood of a formal concept can be formed by finding both the upper and lower neighbours of a formal concept which can be computed separately. In the description of the algorithm that follows, a formal context is denoted by the triplet \(|G, M, I|\) with the finite non-empty sets of objects \(G = \{0,1,\ldots,g\}\) and attributes \(M = \{0,1,\ldots,m\}\) and \(I \subseteq G \times M\) being an incidence relation with \(<g,m> \in I\), meaning that object \(g \in G\) has attribute \(m \in M\). Concept-forming operators defined on \(I\) are denoted by \(\prime : 2^G \mapsto 2^M\) and \(\prime : 2^M \mapsto 2^G\) \([17]\).

The worst-case time complexity of Algorithm 1 is \(O(|G||M|(|G| + |M|))\), the sum of the time to find its lower neighbours, \(O(|G||M|^2)\), and the time to find its upper neighbours, \(O(|G|^2|M|)\). Hence, the maximum running time of the algorithm is quadratic with respect to the number of objects or the number of attributes within the formal context – whichever is larger. As implemented in **CollectionWeb**, the **NearestNeighbours** algorithm runs dynamically at query time – i.e., everytime a user views a formal concept or moves to an upper or lower neighbour, the new concept and its neighbouring concepts are computed. For *ImageSleuth* \([13, 12]\) and *Virtual Museum of the Pacific* case studies \([1]\) this means that any changes to the underlying formal context – new attributes or objects added or removed from the collection – are immediately reflected in its underlying concept lattice, allowing the collection and the relationships among the objects to dynamically respond to user tagging and curatorial management.

However, the advantage offered by dynamically computing the conceptual neighbourhood – namely in that it negates the need to compute or store a potentially large concept lattice while still offering the ability to dynamically expose sections of it for user interaction – also presents another scalability limitation as the size of the collection grows. Given the dynamic nature of the query and the quadratic time complexity with respect to the number of objects in a collection, the *conceptual neighbourhood* approach becomes less suited for use in larger collections, as the response time for user interaction (in the worst case
Algorithm 1: The NearestNeighbours algorithm used for generating
a conceptual neighbourhood for formal concept \(\langle X, Y \rangle\) in formal context \(\langle G, M, I \rangle\), cf. [14]

**Input:** Formal concept \(\langle X, Y \rangle\) of formal context \(\langle G, M, I \rangle\)

**Output:** The set of lower and upper neighbours of \(\langle X, Y \rangle\) in the concept
lattice of \(\langle G, M, I \rangle\)

// Returns the lower neighbours of \(\langle X, Y \rangle\)
lowerNeighbours := \(\emptyset\);
lNCandidates := \(\emptyset\);
foreach \(m \in M \setminus Y\) do
\[ X_1 := X \cap \{m\}'; 
Y_1 := X_1'; \]
if \(\langle X_1, Y_1 \rangle \notin lNCandidates\) then
add \(\langle X_1, Y_1 \rangle\) to \(lNCandidates\);
count(\(\langle X_1, Y_1 \rangle\)) := 1;
else
count(\(\langle X_1, Y_1 \rangle\)) := count(\(\langle X_1, Y_1 \rangle\)) + 1;
if \(|Y_1| - |Y| = count(\(\langle X_1, Y_1 \rangle\))\) then
add \(\langle X_1, Y_1 \rangle\) to lowerNeighbours;

// Returns the upper neighbours of \(\langle X, Y \rangle\)
upperNeighbours := \(\emptyset\);
unCandidates := \(\emptyset\);
foreach \(g \in G \setminus X\) do
\[ Y_2 := Y \cap \{g\}'; 
X_2 := Y_2'; \]
if \(\langle X_2, Y_2 \rangle \notin unCandidates\) then
add \(\langle X_2, Y_2 \rangle\) to unCandidates;
count(\(\langle X_2, Y_2 \rangle\)) := 1;
else
count(\(\langle X_2, Y_2 \rangle\)) := count(\(\langle X_2, Y_2 \rangle\)) + 1;
if \(|X_2| - |X| = count(\(\langle X_2, Y_2 \rangle\))\) then
add \(\langle X_2, Y_2 \rangle\) to upperNeighbours;
scenario) grows quadratically with respect to the number of objects in the collection. While the approach is well suited for dynamically presenting relatively smaller-sized collections at a specialist or ‘exhibition’ sized scale, such as the 427 objects present in the Virtual Museum of the Pacific or the 80 objects present in A Place for Art, the approach remains unsuited for larger collections, such as the the Brooklyn Museum Canvas case study with many thousands of objects.

2.2 The concept layer approach

For all other case studies, COLLECTIONWEB constructs and maintains a set of formal concepts from a formal context of collection objects. The set of all formal concepts for the formal context in COLLECTIONWEB is called the concept layer. The framework relies on a concept layer in order to efficiently create the required data visualisations and semantic structures so that users can associatively browse, visualise and navigate the the collection.

To create and maintain the concept layer, COLLECTIONWEB relies on an algorithm with a low running time for computing formal concepts from a formal context, and for recomputing formal concepts if any objects or attributes in the formal context changes. Specifically, the algorithm should accommodate changes to a formal context in large museum datasets if a single object (or a relatively small batch of objects) changes, ensuring that it can dynamically update the concept layer for a large museum dataset in real time.

There are many high performance algorithms that compute formal concepts from formal contexts [18–22], along with a recent evaluation study of those algorithms applied to data from the Web [23]. As these algorithms offer high performance batch computation of an entire set of formal concepts from a formal context, they work well for large museum collections that do not change over time. However, this is not a common use case: as part of their curatorial practices, museums continually add or modify objects in their online collections, and some require the data to be kept up-to-date as it changes. For instance, the Brooklyn Museum dataset used for the Brooklyn Museum Canvas case study [2], along with other large public facing datasets such as the one provided by the Rijksmuseum 3 – also used in this evaluation – require as part of their terms of use, that all front-facing applications or representation of content must be up-to-date. 4 In these cases, such changes from these data sources should be propagated to these front-facing applications as quickly as possible. In addition, large-scale collaborative tagging efforts such as the steve.museum project [24] and the Flickr Commons recognise museum collections as dynamic, rather than static datasets. As discussed further in Section 2.3, the ability to quickly recompute a set of formal concepts given incremental updates to its formal context can lead to real-time interaction and visualisation of museum data-sets. Such scenarios call for an efficient FCA algorithm that can accommodate incremental

---

3 https://www.rijksmuseum.nl/
4 http://www.brooklynmuseum.org/opencollection/api/docs/terms
changes to a formal context, rather than require the recomputation of the entire set of formal concepts when one or a few of its objects changes.

**CollectionWeb** employs the FCB0 algorithm to initially compute all concepts of a formal context [22] (the algorithm is an improved version of Kuznetsov’s Close-by-One algorithm [25, 26]) and, more importantly, a modification of that algorithm called FCB0 update [27] (earlier version also in [28]) to update formal concepts as objects in the formal context are added, modified or deleted. We briefly present FCB0 update here for the purposes of self-containment. The presentation uses a scenario where new objects are added to the formal context and concept-forming operators that were used in Algorithm 1. In addition, new objects to be added to \( \langle G, M, I \rangle \) and not present in \( G \) are denoted by \( G_N = \{ g + 1, \ldots , g_U \} \) (i.e. \( G_N \cap G = \emptyset \)), \( M_N = \{ i, \ldots , k \} \) is the set of attributes shared by at least one of the objects \( G_N \) and either present or not present in \( M \) (but usually \( M_N \subseteq M \)) and \( N \subseteq G_N \times M_N \) is an incidence relation between \( G_N \) and \( M_N \). By the triplet \( \langle G_U, M_U, I_U \rangle \) we denote the formal context which results as a union of \( \langle G, M, I \rangle \) and \( \langle G_N, M_N, N \rangle \), both extended to \( G_U \) and \( M_U \), i.e. \( G_U = G \cup G_N = \{ 0, \ldots , g_U \} \), \( M_U = M \cup M_N = \{ 0, \ldots , m_U \} \), \( m_U = k \) if \( k > m \) and \( m_U = m \) otherwise, and \( I_U \subseteq G_U \times M_U \) such that \( I_U \cap (G \times M) = I \), \( I_U \cap (G_N \times M_N) = N \) and \( I_U \cap (G \times (M \setminus M_N)) = I_U \cap (G_N \times (M \setminus M_N)) = \emptyset \).

The algorithm is represented by the recursive procedure \texttt{UpdateFastGenerateFrom}, presented in Algorithm 2. The procedure is a modified form of the recursive procedure \texttt{FastGenerateFrom} – the core of the FCB0 algorithm as described in [22] (Algorithm 2). The procedure accepts as its arguments a formal concept \( \langle X, Y \rangle \) of \( \langle G_U, M_U, I_U \rangle \) (an initial formal concept), an attribute \( m \in M_N \) (first attribute to be processed) and a set \( \{ N_m \subseteq M_U \mid m \in M_U \} \) of subsets of attributes \( M_U \), and uses a local variable \texttt{queue} as a temporary storage for computed formal concepts and \( M_m (m \in M_U) \) as sets of attributes which are used in place of \( N_m \) for further invocations of the procedure. When the procedure is invoked, it recursively descends, in a combined depth-first and breadth-first search, the space of new and updated formal concepts of \( \langle G_U, M_U, I_U \rangle \) resulted by adding new objects \( G_N \) described by attributes \( M_N \) to \( \langle G, M, I \rangle \), beginning with \( \langle X, Y \rangle \). For a full description of the procedure, see [27] or [28], recalling that the set \( M_{U,j} \subseteq M_U \) in Algorithm 2 is defined by: \( M_{U,j} = \{ m \in M_U \mid m < j \} \). In order to compute all new and updated formal concepts of \( \langle G_U, M_U, I_U \rangle \) which are not formal concepts of \( \langle G, M, I \rangle \), each of them exactly once, \texttt{UpdateFastGenerateFrom} shall be invoked with \( \langle \emptyset, \emptyset’ \rangle, m \) being the first attribute in \( M_N \) and \( \{ N_m = \emptyset \mid m \in M \} \) as its initial arguments.

The worst-case time complexity of Algorithm 2 remains the same as of the original FCB0 (and CbO) algorithm, \( O(|C||M|^2|G|) \), because when adding all objects to the empty formal context it actually performs FCB0.

For updating a set of formal concepts given by incremental object-by-object updates of a formal context, there are a number of other incremental algorithms that can be used for determine a set of formal concepts and, subsequently, for
Algorithm 2: The UpdateFastGenerateFrom(⟨X, Y⟩, m, {Nm | m ∈ MU}) algorithm used for computing all new and updated formal concepts of formal context ⟨GU, MU, IU⟩, cf. [27]

Input: Formal concept ⟨X, Y⟩ of formal context ⟨GU, MU, IU⟩, attribute \( m \in MN \) (or a number \( \geq m_U \)) and set \{Nm ⊆ MU | m ∈ MU\} of subsets of attributes \( MU \)

Output: The set of all new and updated formal concepts of ⟨GU, MU, IU⟩

// output ⟨X, Y⟩, e.g., print it on screen or store it
if \((X ∩ G)′ \neq Y\) then
  output ⟨X, Y⟩ as new;
else
  if \((X ∩ G) \subset X\) then
    output ⟨X, Y⟩ as updated;
  else
    return
if \(Y = MU\) or \(m > m_U\) then
  return
for \(j\) from \(m\) upto \(m_U\) do
  set \(M_j\) to \(N_j\);
  // go through attributes from \(MN\) only
  if \(j \notin Y\) and \(j \in MN\) and \(N_j ∩ MU,j \subseteq Y ∩ MU,j\) then
    set \(X_1\) to \(X \cap \{j\}\);
    set \(Y_1\) to \(X_1′\);
    if \(Y ∩ MU,j = Y_1 ∩ MU,j\) then
      put \(⟨⟨X_1, Y_1⟩, j + 1⟩\) to queue;
    else
      set \(M_j\) to \(Y_1\);
  while get \(⟨⟨X_1, Y_1⟩, j⟩\) from queue do
    UpdateFastGenerateFrom(⟨X1, Y1⟩, j, {Mm | m ∈ MU});
return
computing the concept lattice, such as [16, 29, 30] along with the algorithms in [14]. AddIntent [30] is considered to be one of the most efficient of these algorithms, however, along with the other algorithms, it requires the complete concept lattice prior to computation. The FCB update algorithm [27] described above, differentiates itself from other incremental algorithms in that it does not require the concept lattice (nor the set of all formal concepts) as its input. However, the number of concepts computed from datasets we use – even without the complexities of storing a complete concept lattice – is of the order hundreds of thousands (see Figures 1 and 2). In light of this, the FCB update algorithm not only computes changes based only on a set of objects marked for update, but it also outputs only the new and updated formal concepts, rather than the entire set of formal concepts. This allows for quick execution of the algorithm and ingestion of its results where changes to formal context are relatively minor: strengthening the algorithm’s utility in applications where datasets are large but updated frequently and in small increments.

2.3 Performance Evaluation

The algorithm was evaluated on two museum datasets: the first being the Brooklyn Museum collection consisting of 10,000 objects and 8,952 attributes and the second being the Rijksmuseum collection consisting of 100,000 objects and 1,716 attributes. The purpose of the performance evaluation was to determine the total running time and performance benefit of using the FCB update algorithm to incrementally update a set of formal concepts given changes to a formal context, rather than recomputing its entire set of formal concepts.

Table 1. Running time of computing all formal concepts from a formal context using the FCB update algorithm, average of 10 iterations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. of attributes</th>
<th>No. of objects</th>
<th>No. of concepts</th>
<th>Avg. running time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brooklyn Museum</td>
<td>8,952</td>
<td>10,000</td>
<td>98,547</td>
<td>36,218</td>
</tr>
<tr>
<td>Rijksmuseum</td>
<td>1,716</td>
<td>100,000</td>
<td>994,967</td>
<td>68,792</td>
</tr>
</tbody>
</table>

Table 1 shows the running time to compute the entire set of formal concepts from the formal contexts generated from the Brooklyn Museum and Rijksmuseum datasets. For the sake of clarity, a batch or non-update computation – such as the one demonstrated in the table above – is defined as a computation that computes the entire set of formal concepts from formal context, whereas an update formal concept computation is defined as a computation that uses a set of objects to add, remove or update within the formal context as its input and outputs a set of changed concepts. The above figures in Table 1 are used as a benchmark in the evaluation of the performance benefit of the update, rather than the batch computations of the FCB update algorithm.
An update computation can be triggered by three different events: adding new objects to the formal context, removing existing objects from the formal context, or updating the attribute sets of existing objects within the formal context. Given that objects can be added, removed or updated within a museum dataset, these three operations are defined and evaluated separately with respect to the running time of the algorithm. Assuming a full set of formal concepts have already been computed, each operation produces a number of modified concepts that refer to the set of formal concepts added, removed or updated as a result of each operation. In addition to the time it takes to perform each operation, the number of modified concepts serves as an important indicator of complexity.

The results of a performance evaluation demonstrating add, remove and update operations for the FCbO update algorithm are shown in Fig. 1 for the Brooklyn Museum dataset, and Fig. 2 for the Rijksmuseum dataset. The figures demonstrate how the algorithm scales with each operation for adding, removing or updating 1, 5, 50 or 500 objects to their respective datasets. In each figure, the horizontal axis first groups the number of objects \( N \), which is then further sub-divided into its three operations with respect to the formal context: incrementally compute the set of formal concepts when \( N \) objects are added, removed and updated from the formal context. As a way of comparing the running time of the FCbO update algorithm to its batch counterpart, the performance metrics of the update algorithm – its running time and number of modified concepts – are shown along with the total running time and number of formal concepts produced by the non-update algorithm, the dashed line in Figures 1 and 2.

For the smaller Brooklyn Museum collection, the number of modified concepts and time taken to compute them is reasonable when adding 5 or 50 objects, with running times far less than the time it takes for the algorithm to recompute the entire set of formal concepts. However, in the larger Rijksmuseum collection – due to the smaller number of attributes and higher context density – removing and updating a larger batch of objects requires the re-computation of a large number of formal concepts where in some cases, Figures 1 and 2, the time taken to update the set of formal concepts is greater than the time to recompute the entire set as a batch operation.

The benefits of an incremental FCbO update algorithm with a low running time with respect to museum curation practices and visitor experiences can be realised with respect to user interactions that lead to dynamically changing contexts. For example, in many online collections such as the Powerhouse Museum Online Collection and the Brooklyn Museum Online Collection, visitors can add their own interpretations to the objects by adding their own keywords or ‘tags’. These interactions can introduce new perspectives on the works that can potentially reframe the way objects are related to one another in that audiences are invited to shape the context, and subsequently, the knowledge that surrounds the objects. Given that formal concepts can be used to represent contextual knowledge of a domain where museum objects are treated as formal

\[ \text{http://www.powerhousemuseum.com/collection/database/menu.php} \]
\[ \text{https://www.brooklynmuseum.org/opencollection/collections/} \]
Fig. 1. Average running time and number of modified concepts for adding, removing or updating objects to a formal context and incrementally recomputing the set of formal concepts using the FCbO update algorithm on the Brooklyn Museum dataset. The top graph shows the total running time for each operation for 1, 5, 50 and 500 objects, whereas the bottom graph shows the total number of modified concepts for each operation for 1, 5, 50 and 500 objects.
Fig. 2. Average running time and number of modified concepts for adding, removing or updating objects to a formal context and incrementally recomputing the set of formal concepts using the FCbO update algorithm on the Rijksmuseum dataset. The top graph shows the total running time for each operation for 1, 5, 50 and 500 objects, whereas the bottom graph shows the total number of modified concepts for each operation for 1, 5, 50 and 500 objects.
objects and tags as formal attributes, user tagging can provide the ability to update representations of knowledge in real-time. Due to the low running time of the FChO update algorithm on small sets of objects as their input, a user could potentially tag an object and then, through the use of incremental concept computation coupled with data visualisation, immediately realise not only how their tagging enhances the content of the objects, but also shapes the knowledge that surrounds it in relation to other objects.

In many other cases, updates to museum collection data are provided as a batch – i.e., whole groups of objects added or modified as a result of changes to objects within a museum dataset. For example, the Smithsonian Cooper-Hewitt National Design Museum uses GitHub\(^7\) to host their collection data\(^8\) – allowing anyone to access, update and provide updates to the collection. Many other museums provide a timestamp in their object records to indicate when it was last updated, so that data harvesters can collect changes. In other situations it may be more feasible to implement updates to the dataset as a batch rather than as a set of small frequently occurring object updates.

### 3 Conclusion

Overall, the FChO update algorithm – as implemented by CollectionWeb to construct and maintain its concept layer – provides a fast way to update formal concepts from large and dynamically changing museum datasets, given that the changes within those datasets are relatively small relative to the size of the formal context. The algorithm provides a scalable way to construct and maintain a concept layer once the initial and potentially time costly computation of the entire set of formal concepts from a formal context is complete. The algorithm is less efficient at adding, removing or updating large changes to the collection where, in such cases, it may be preferential to recompute the entire set of formal concepts.

### References


\(^7\)GitHub is a popular source code management system traditionally used for making available, committing and providing updates to, program source code.

\(^8\)See: [http://www.cooperhewitt.org/collections/data](http://www.cooperhewitt.org/collections/data)