Abstract—Multimedia Indexing and Retrieval is generally designed and implemented by employing feature graphs. These graphs typically contain a significant number of nodes and edges to reflect the level of detail in feature detection. A higher level of detail increases the effectiveness of the results but also leads to more complex graph structures. However, graph-traversal-based algorithms for similarity are quite inefficient and computation intensive, especially for large data structures. To deliver fast and effective retrieval, an efficient similarity algorithm, particularly for large graphs, is mandatory. Hence, in this paper, we define a graph-projection into a 2D space (Graph Code) as well as the corresponding algorithms for indexing and retrieval. We show that calculations in this space can be performed more efficiently than graph-traversals due to a simpler processing model and a high level of parallelisation. In consequence, we prove that the effectiveness of retrieval also increases substantially, as Graph Codes facilitate more levels of detail in feature fusion. Thus, Graph Codes provide a significant increase in efficiency and effectiveness (especially for Multimedia indexing and retrieval) and can be applied to images, videos, audio, and text information.

Keywords—indexing, retrieval, multimedia, graph code, graph algorithm

I. INTRODUCTION AND MOTIVATION

Multimedia assets like images, videos, texts, or audio are deeply integrated in today’s life for many users. The ease of creating Multimedia content e.g., on Smartphones, and publishing it on Social Media is unseen in history. Infrastructure services like high-speed networks, cloud-services, or online storage need a good and fast indexing of Multimedia content [1] as e.g. every single minute, 147.000 photos are uploaded to Facebook, 41.6 million WhatsApp messages are sent, or 347.000 stories are posted by Instagram [2]. Semantic indexing and fast retrieval of these assets are essential for managing this large amount of information. For this task it is common to use graph-based technologies and structures, as Multimedia information is based on feature nodes and links between these nodes [3]. To increase the retrieval accuracy, increased data from various sources (e.g. Social Media, Documents, Semantic Web, embedded metadata) is fused into large feature-graphs. But to employ these features particularly for precise retrieval, fast graph-based similarity algorithms are required. Current solutions, as e.g. Neo4J databases [4] and their integrated algorithms fail to deliver acceptable processing times for retrieval.

In this paper, we describe fast and accurate Indexing and Retrieval algorithms for Multimedia feature-graphs. These algorithms are based on a projection of graphs into a 2D space, which supports parallelisation in retrieval and can utilize standard pattern matching algorithms including Machine Learning. The presented concept can be applied to any type of Multimedia feature, which can be represented as a graph (e.g. images, videos, text, audio). In our experiments we will give detail on performance, accuracy and quality based on various datasets, measured on various devices including tablets. In section II, we summarize the current state of the art and related work. Section III gives the mathematical and algorithmic details of Graph Codes and their application, which are implemented in section IV. Section V discusses detailed results of experiments regarding effectiveness and efficiency. Finally, in section VI a conclusion is summarized and an outlook to future work is given.

II. STATE OF THE ART AND RELATED WORK

This section provides an overview of current Multimedia feature extraction techniques supporting indexing and retrieval, which either represent or contribute to indexing features of Multimedia content. Selected indexing and retrieval of textual information, images, audio, and video assets is described including current concepts for feature fusion, which will result in large semantic graph structures. These technologies can all contribute information to feature graphs. A brief description of the mathematical background of these graphs is given including an overview of current datasets for indexing and retrieval.

In our previous work, we already introduced the Generic Multimedia Analysis Framework (GMAF) [5], [6], [7], [8] as an unifying framework, that is able to fuse various Multimedia features into a single data structure. The GMAF utilizes selected existing technologies as plugins to support various Multimedia feature detection algorithms for text (e.g. social media posts, descriptions, tag lines) [9], [10], [11], images (especially object detection and spatial relationships including the use of machine learning) [12], [13], [9], [14], [9], audio (transcribed to text) [15], [16], [13], and video including metadata [17] and detected features [18], [19], [16]. In general, every detected feature can be regarded as a Multimedia indexing term. The indexing term of any relevant feature thus becomes part of the vocabulary of the overall retrieval index. In Multimedia Indexing and Retrieval (MMIR), these terms typically have structural and/or semantic relationships to each other. Thus, graph-based structures are appropriate candidates to represent Multimedia features including their structural and
A Feature Graph (MMFG) defined as semantic and technical features of Multimedia Assets and is represented by types. The MMFG is a weighted and directed graph and fuses semantic relationships. The GMAF provides an extendable representation schema and processing architecture for fusing detected Multimedia features and generating Multimedia Feature Graph (MMFG) data structures. The Multimedia Feature Graph (MMFG) is a graph structure for representing semantic and technical features of Multimedia Assets and is defined as $MMFG_{Asset} = (N, E)$, where $N$ is the set of nodes and $E$ the set of edges between these nodes. Both $N$ and $E$ are employed to represent special Multimedia Features and their relationship, that have been detected within an asset (e.g. instances of object, region, colour, or relationship features). Elements of $N$ and $E$ are represented by types. The MMFG is a weighted and directed graph and fuses technical and semantic information into a single model [20]. A complete description of the MMFG is given in [5], a reference implementation is available on Github [7], and a visualisation of a small section of an exemplary MMFG is shown in Figure 1a, which shows and exemplary MMFG including several feature types in different node and edge type colours (e.g. detected objects in blue, detected landmarks in yellow, synonyms in green, spatial relationships in red). Visualisation has been performed with yEd [21]. A complex MMFG contains feature representations e.g. from text (e.g. metadata or Social Media), images (e.g. objects, colours, spatial attributes), video, and audio information (if applicable) and Figure 1a shows an exemplary MMFG snippet, where the following feature categories are visible: object detection, dominant colours, spatial relationships, landmark detection.

From a Mathematical Perspective, graphs can be represented through their Valuation Matrices, which extend Adjacency Matrices and integrate the information of the weights of edges [22], also enabling the application of mathematical concepts to graphs [23]. Similarity calculation on Valuation Matrices can be performed with the Eigenvalue Method [22]. However, each mathematical approach usually has a complexity of $O(n^2)$ ($n$ nodes and $e$ edges). Several approaches are described to improve this for a particular set of data or within particular conditions [24], [25], [26], but the performance of these algorithms for large data structures, like feature graphs of Multimedia objects, still has to be improved. In the remainder of this paper we describe and evaluate the Graph Code Encoding Algorithm, Query Construction, and Retrieval Execution with Graph Codes, as an extension of graph-based Valuation Matrices for MMIR applications. These algorithms can perform comparison tasks based on graph data in $O(n + e) + O(1)$.

W.r.t Graph Codes and their Encoding, the following discusses the mathematical and algorithmic concepts of a 2D graph representation and its relevance for MMIR. We have introduced Graph Codes in [5]. They are applicable to any kind of graph, but we specifically designed them for MMIR. We employ them to represent MMFGs and to transform these into another mathematical space for fast and efficient MMIR. Our Graph Code Encoding algorithm [5] uses the Valuation Matrix $VM$ of a given MMFG as a basis. This approach is expected to require fewer calculations than e.g., comparable vector-space transformations. As an example, we employ a small subgraph of the MMFG of Figure 1b containing the most relevant structures to illustrate the concepts presented in this paper. Thus, we define our example-graph $MMFG_{ex}$ as shown in Figure 1b.

Valuation Matrices contain 1 row and column for each node always resulting in square matrices. Edges between nodes $n_1$ and $n_2$ are represented in the matrix with their weight or a value of 1 at position $(n_1, n_2)$. For the above example, the set of nodes $N$ is given by $N = \{Person, Head, Human Being, Individual, Hat, above\}$, represented by a value of 1 in one of the diagonals of the matrix. Thus, the Valuation Matrix $VM$ is defined as shown in Figure 1c. Graph Codes employ an encoding function $f_{enc}$, which calculates a numeric value for each non-zero position of a Valuation Matrix based
on node or edge type and the corresponding attributes. If we apply such a function to the above example, the encoded Valuation Matrix $VM_{enc}^f$, i.e., the corresponding Graph Code $GC$ is shown in Figure 1d. Going beyond this encoding, other information about nodes and edges can be encoded with more complex functions resulting in arbitrary natural numbers. Based on Graph Codes, we introduce algorithms for Query Construction and Query Execution in section III, which are later evaluated in sections IV and V. To prove their effectiveness and efficiency, comprehensive and well annotated sample data is required.

To evaluate these Multimedia retrieval algorithms, an appropriate Annotated Sample Dataset has to be available. As the evaluation has to prove effectiveness, efficiency and quality, the sample dataset must contain content description annotations, a high Level Of Detail (LOD), and a relevant number of samples to be processed. To analyse algorithms for Multimedia processing, several datasets can be employed. One of the most comprehensive collections of annotated text-based sample data is maintained by the Text Retrieval Conference (TREC) [27], a detailed overview of Audio datasets (e.g. The Spoken Wikipedia Corpora) is given by [28] and a commonly used dataset for video processing is the Youtube8M [29]. For image processing, the Flickr30k set [30], the DIV2K dataset [31], the IAPRTC12 dataset [32], or the PASCAL VOC dataset [33] are some of the most relevant collections. In our evaluation, we initially focus on image processing, as feature extraction of images provides a high level of detail and the sample datasets provide data suitable for the experiments. Thus, high-resolution datasets with accurate annotations are required to perform a recursive feature extraction and to measure efficiency comparisons of algorithms. Hence, we selected the Flickr30k, Div2K, and the PASCAL datasets.

**In summary**, we can state that current technologies provide a sufficient set appropriate algorithms, tools, and concepts for extracting features of Multimedia content. Integrating data structures as, e.g., the MMFG can fuse this information and compile it into a large semantic graph structure. However, the need to fuse many features into graphs to increase effectiveness contradicts the demand for higher performance for retrieval, as graph-traversal algorithms become less efficient with an increasing number of nodes and edges. Hence, finding a solution that both provides a high level of detail for effective retrieval and a highly performant and efficient model for similarity algorithms is one major open challenge. This includes the application of the Graph Code encoding to MMIR, the selection or preparation of an appropriate test collection, and evaluation of the solution.

---

3The function $f_{enc}$ can be adjusted deliberately to meet the requirements of any application. In this paper, we chose an arbitrary selection of value ranges representing type attributes of nodes and edges for the Graph Code representation.

2The same coloring as in Figure 1b is applied. Values for types are selected as follows: object-node = 1, synonym-node = 2, child-relationship = 3, synonym-relationship = 4, relationship = 5, spatialrelationship-node = 6. Node representing fields are given in bold.

### III. Modeling and Design

A typical information retrieval function or algorithm $IR$ for a given query $Q$ and a result set $R$ can be generally defined as $IR(Q) \rightarrow R$. Here, $Q$ is represented by a $MMFV_{Query}$ object representing the query-features, and $R$ is a ranked list of MMFGs. The retrieval function $IR$ calculates the relevance based on the similarity between $MMFV_{Query}$ and each element of the set of existing $MMFG$s. For graphs like the MMFG, a metric for similarity would be e.g., the Cosine Similarity [25]. Thus, for MMFGs, the retrieval function is defined as $IR_{MMFG}(MMFV_{Query}) = \{MMFG_1, ..., MMFG_n\}$. In the case of Graph Codes and the corresponding algorithms, each MMFG is represented by its $GC_{MMFG}$ and the retrieval function is $IRGC(GC_{Query}) = (GC_1, ..., GC_n)$. The result of $IRGC$ is a ordered vector of all Graph Codes of the collection, in which $VCG_i \in IRGC : GC_i > GC_{i+1}$. The comparison of Graph Codes has to be based on a well-defined metric for similarity, in which both the mathematical aspects of matrix comparison, and the semantic aspects of Graph Code representation must be considered.

Thus, we now define **Graph Code Feature Vocabularies and Dictionaries** that are needed for MMIR. Based on the definitions, a metric for similarity calculations going beyond node and edge types, can be defined. In each MMFG, the set of $n$ nodes representing distinct feature terms can be regarded as unique identifiers for the MMFG’s feature term vocabulary $FVT_{MMFG} = \{f_{t1}, ..., f_{tn}\}$. This set of a MMFG’s vocabulary terms thus represents the elements of a corresponding Graph Code’s Dictionary, i.e. the set of all individual feature vocabulary terms of a Graph Code. However, it is important to uniquely identify the feature vocabulary term assigned to a field of a Graph Code. Thus, we introduce a Graph Code Dictionary for each Graph Code, which is represented by a vector $dict_{GC}$ and provides a ordered representation of the set $FVT_{MMFG}$ with uniquely defined positions for each MMFG’s feature vocabulary term. The elements in $dict_{GC}$ can be ordered according to the corresponding MMFG. In the Graph Code matrix representation, each node field (in the diagonal) of a Graph Code can now be unambiguously mapped to an entry of its Graph Code Dictionary vector, which can be represented as $dict_{GC} = (f_{t1}, ..., f_{tn})$. Applied to the Graph Code of the previous example, the set of feature vocabulary terms $FVT_{ex}$ would be $\{Person, Head, Human Being, Individual, Hat, above\}$, in which the elements do not have any order. The corresponding vector $dict_{ex} = (Person, Head, Human Being, Individual, Hat, above)$ and - in difference to the set representation - uniquely identifies each vocabulary term by its position within the vector. When comparing the similarity of 2 Graph Codes, it is important to compare only feature-equivalent node fields in the diagonal of each matrix to each other. Each Graph
Code has its own, individual dictionary-vector \textit{dict}_\text{GC}, and another \textit{Graph Code} will have a different dictionary-vector according to the content of its represented MMFG, typically \textit{dict}_{\text{GC1}} \neq \textit{dict}_{\text{GC2}}. Feature-equivalent node fields of \textit{Graph Codes} can be determined through their corresponding \textit{Graph Code Dictionaries}, as these fields will have positions represented by an equal feature vocabulary term of each corresponding dictionary. For comparison, only the set of intersecting feature vocabulary terms of e.g. 2 \textit{Graph Codes} is relevant. Thus, the set of intersecting feature vocabulary terms \textit{FVT}_{\text{Coll}} of e.g. 2 MMFGs can be defined as \textit{FVT}_{\text{Coll}} = \{f_{t1}, ..., f_{tn}\} = \textit{VMMFG}_1 \cap \textit{VMMFG}_2. The methodology of intersecting sets can be also applied to \textit{Graph Code} dictionaries. The intersection of two vectors \textit{dict}_{\text{1,2}} can be defined correspondingly as \textit{dict}_{\text{1,2}} = \textit{dict}_{\text{GC1}} \cap \textit{dict}_{\text{GC2}}. To illustrate the calculation of \textit{dict}_n, we introduce a second exemplary \textit{Graph Code} \textit{GC}_{ex2} based on a \textit{MMFG}_{ex2}, which is shown in Figure 2a.

The set \textit{FVT}_{\text{ex2}} in this case is \{\textit{above}, Dog, Head, Animal, Hat\} and the set \textit{FVT}_{\text{Coll}} of intersecting feature vocabulary terms is \{\textit{above}, Head, Hat\}. The dictionary-vector \textit{dict}_{\text{ex2}} thus is \textit{dict}_{\text{ex2}} = \{\textit{above}, Dog, Head, Animal, Hat\}. Figure 2b shows its table representation and Figure 2c its list representation \textit{dict}_{\text{ex2}}. The vector \textit{dict}_{\text{ex1,2}} represents the dictionary of intersecting vocabulary terms and only contains the subset of vocabulary terms of \textit{dict}_{\text{ex2}}, where a valid vocabulary term can be found in \textit{dict}_{\text{ex2}}. The order of intersecting vocabulary terms in \textit{dict}_{\text{ex1,2}} is given by the order of \textit{dict}_{\text{ex2}}. From an algorithmic perspective, this means, that all elements of \textit{dict}_{\text{ex2}} are deleted, that cannot be found in \textit{dict}_{\text{ex2}}. The index position of \textit{dict}_{\text{ex1,2}} typically is different from both \textit{dict}_{\text{ex1}} and \textit{dict}_{\text{ex2}}. Based on these dictionary-vectors, a translation of equivalent \textit{Graph Code} positions can be performed, as each feature vocabulary term has a unique position within each of the \textit{Graph Code}’s dictionaries.

Applications will typically utilize a collection of MMFGs and their corresponding \textit{Graph Codes}. The overall feature term vocabulary \textit{FVT}_{\text{Coll}} = \{f_{t1}, ..., f_{tn}\} containing \textit{c} vocabulary terms of such a collection of \textit{n} MMFGs can be defined as the union of all MMFG’s feature term vocabularies and also be represented by the union of all \textit{Graph Code Dictionaries} \textit{dict}_{\text{j}}: \textit{FVT}_{\text{Coll}} = \bigcup_{j=1}^{n} \textit{FVT}_{\text{MMFG}_j}, where \forall i,j < n : \textit{dict}_{\text{j}} = \textit{dict}_{\text{i}} \times \textit{dict}_{\text{j}}. In this \textit{dict}_{\text{j}}, dictionary-union-vector, the \times-operation for calculating the union of dictionary-vectors is implemented by traversing all the collection’s \textit{dict}_{\text{j}} dictionaries and collecting unique dictionary vocabulary terms into a single dictionary-vector. In our example with \textit{dict}_{\text{ex1}} and \textit{dict}_{\text{ex2}}, the calculated \textit{dict}_{\text{ex}} = (\text{Person, Head, Human Being, Individual, Hat, above, Dog, Animal}). If a \textit{dict}_{\text{ex}} is calculated for the complete collection of \textit{Graph Codes}, it can be regarded as a global dictionary-vector with collection-wide unique positions for each feature vocabulary term.

Processing many different MMFGs will result in many different \textit{Graph Codes} having similar sizes, but different vocabulary terms, leading to an increase of \textit{V}_{\text{Coll}}. The Oxford English Dictionary \cite{34} e.g., contains 170.000 english words\footnote{Translation and multilingual support is not in scope of this paper and does not affect the general concept of \textit{Graph Codes}.} and if we assume, that applications exist, which produce english terms as representations for feature-nodes, MMFGs representing the overall vocabulary would result in matrices of size \textit{170.000} \times \textit{170.000} giving 28.9 billion matrix fields. Calculations on this large number of fields will no longer be efficient enough for MMIR. Of course, in some use cases, an application-wide dictionary can be required. But in some other applications, it would be prudent to employ a smaller dictionary. Hence, two major approaches of modeling dictionaries can be proposed:

\textbf{Application-wide dictionary:} In this scenario, we assume that any \textit{Graph Code} will be processed with the dictionary-vector terms \textit{dict}_{\text{i,j}}. If in an MMIR application all images are almost similar, a processing and re-processing approach can automatically increase or decrease the collection’s vocabulary terms according to the analysis of new content. All existing \textit{Graph Codes} have to be adjusted whenever new indexing terms are detected (the size of \textit{dict}_{\text{i,j}} increases) or whenever existing Multimedia feature content is removed from the collection (the size of \textit{dict}_{\text{i,j}} decreases). The big advantage of this approach is, that all \textit{Graph Codes} have exactly the same size and identical positions represented by their dictionary-vectors. This makes comparisons very easy as no further transformation is required. It also simplifies the employment of Machine Learning algorithms. However, a permanent re-processing of all existing \textit{Graph Codes} can be very expensive. In this case, the following scenario should be preferred.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig2.png}
\caption{MMFG\textsubscript{ex2} and representations. Figure 2a shows a second exemplary MMFG, 2b the corresponding Graph Code \textit{GC}_{ex2}. Figure 2c illustrates the dictionary of \textit{GC}_{ex2}, and 2d the intersections \textit{M}_{\text{i}}. Figure 2e illustrates the subtraction, 2f the encoded intersection.}
\end{figure}
Dictionaries for smaller combinations of individual vocabularies: if images from many different areas (i.e. with many different feature vocabulary terms) have to be processed, two Graph Codes can be compared based on the intersection of their individual Graph Code’s dictionary vectors dicti. In this case, a mapping of corresponding feature vocabulary terms by their position within each dictionary-vector can be performed and equivalent node matrix fields can be calculated by a simple transformation (i.e. re-ordering) of one of the dictionary-vectors. As this also eliminates many unnecessary operations (e.g. comparing unused fields of dicti), this approach can be very efficient, when Graph Codes vary significantly within a collection.

Applied to GCcx and GCcx2 of our example above, the application-wide dictionary dictcx would give Graph Codes with a size of 9 × 9 matrix fields, whereas dictcx2 would give an intersection matrix of 3 × 3 fields. This intersection matrix M(GC) can be calculated from a GC by removing any rows and columns, that are not part of dictcx2. Figure 2d shows the intersection matrices of GCcx and GCcx2.

For comparison of these intersection matrices, we wish to apply the standard matrix subtraction. However, due to the different orders of dictcx and dictcx2, the matrix field positions of the matrices do not represent the same feature vocabulary terms. For example, the field (2,1) of GCcx represents the relationship between Hat and Head, but the equivalent relationship in GCcx2 is located in field (3,2). To solve this, we introduce a equivalence function feqv(M), which transforms a Graph Code intersection matrix or the corresponding dictionary-vector in such a way, that the corresponding dictionary-vector is ordered according to dictcx2.

Thus, equivalence of a matrix field (mi,j) in M(GCcx) and a matrix field (nj,k) in M(GCcx2) and corresponding dictionary vectors dictcx and dictcx2 can be defined as: ∀(mi,j) ∈ M(GCcx), ∀(nj,k) ∈ M(GCcx2):  

\[ M(GCcx) \sim feqv(M(GCcx2)) \Leftrightarrow \]

\[ dictcx(i) = feqv(dictcx2(j)) \land dictcx(j) = feqv(dictcx2(i)) \]

In the case of comparing only 2 Graph Codes, dictcx2 automatically is ordered according to the first Graph Code. Thus, in this case, the second dictionary-vector would be re-ordered to match the order of the first one. This re-ordering is also applied to the corresponding intersection matrix of the Graph Code. In case of our example, dictcx2 = (above, Head, Hat) would be re-ordered to match dictcx = (Head, Hat, above). Thus, the resulting re-ordered intersection matrix is shown in Figure 2e.

Based on the results of this section, we now define a metric to calculate Graph Code Similarity as a basis for MMIR retrieval applications. This metric enables MMIR application to compare Graph Codes and thus utilize them for retrieval. In case of Graph Codes and their matrix-based representation, the calculation of similarity requires the consideration of rows, columns and fields representing nodes and edges (i.e. node relationships) of a MMFG. These nodes and relationships have to be of equivalent node or relationship type for comparison.

This means, that it is important to compare the correct matrix field position to each other, which typically is different in e.g., 2 Graph Codes. Matrix field positions employed for the definition of a metric represent nodes (i.e. detected features) or edges (i.e. detected node-relationships), edge-types (i.e. detected node relationship types), and their type values. The definition of a metric for Graph Codes has to be applicable for matrices, where rows and columns represent MMFG nodes and the corresponding feature vocabulary terms. Matrix cells represent node types (in one diagonal) and all other non-zero matrix fields represent edge types and their values.

Based on these characteristics of Graph Code we can define a metric as a triple of metrics containing a feature-metric MF, a feature-relationship-metric MFＲ and a feature-relationship-type-metric MFＲT.

The Graph Code Feature Metric MF can be employed to calculate the similarity of Graph Codes according to the intersecting set of dictionary vocabulary terms. MF is defined as the ratio between the cardinality of dicti, the intersecting dictionary vocabulary terms and the cardinality dicti, a Graph Code’s dictionary vector. In the following formulae, the notation |v| for vectors denotes the cardinality of a vector v, i.e. the number of elements in this vector:  

\[ MF(GC1, GC2) = \frac{|dicti|}{|dicti|} \]

Thus, the more features are common in e.g., 2 MMFGs, the higher the similarity value based on MF - independent of the relationships between these corresponding MMIR features. In the above example, the numerical distance between GCcx and GCcx2 based on the metric MF is  

\[ MF(GCcx, GCcx2) = \frac{|dictcx|}{|dictcx2|} = \frac{3}{6} = 0.5. \]

The Graph Code Feature Relationship Metric MFＲ is the basis for the similarity calculation of MMFG-edges, i.e. the non-diagonal and non-zero fields (representing edges of deliberate types) of the Graph Code’s matrix representation.
This metric is only applied to equivalent fields (i.e. relationships with the same source and target node) of intersection matrices $M_I$ of two Graph Codes. We base this metric on the non-diagonal fields of the Adjacency Matrix $AM(M_I)$ (i.e. the matrix containing only the values 1 and 0). Then, $M_{FR}$ can be defined as ratio between the sum of all non-diagonal fields and the cardinality of all non-diagonal fields.

$$M_{FR}(GC_i, GC_j) = \frac{\sum AM(M_{i,j}) - n}{|AM(M_{i,j})| - n}$$

Thus, in terms of our example, the overall similarity $M_{GC}$ between $GC_{ex}$ and $GC_{ex2}$ is $M_{GC}(GC_{ex}, GC_{ex2}) = (M_{FR}, M_{RT}) = (0.5, 0.33, 0.16)$. This means, that the similarity based on common vocabulary terms $M_{FR}$ is 0.5, the similarity based on common edge positions $M_{RT}$ is 0.33, and the similarity of equal edge types $M_{RT}$ is 0.16.

Based on these metrics for Graph Codes, the MMIR retrieval can utilize comparison functions to calculate a ranked list of results. **Query Construction with Graph Codes** is possible in 3 ways: a manual construction of a query `Graph Code GC_query`, the application of the `Query by Example` paradigm, or an adaptation of existing Graph Codes. A manual construction of a MMFG by users can result in a `GC_query Graph Code`, which then is employed for querying. This manual construction could be performed by entering keywords, structured queries (e.g. in a query language like SPARQL [35]), or also natural language based commands [36] into a MMIR application’s query user interface. The MMFG and corresponding `GC_query` in this case is created completely from scratch. Query construction can be based on the `Query by Example` paradigm [37]. In this case, a `GC_query` is represented by an already existing `Graph Code`, which typically is selected by the user to find similar assets in the collection of a MMIR application. An adaptation of an existing `Graph Code` can lead to a `GC_query` as well. A refinement in terms of `Graph Codes` means, that e.g. some non-zero fields are set to zero, or that some fields get new values assigned according to the `Graph Code` encoding function $f_{enc}$. From a user’s perspective, this can be performed by selecting detected parts of corresponding assets and selecting, if they should be represented in the query or not. A prototype for all 3 options of `Graph Code` querying is illustrated in [5] and available on Github [7]. The adaptation of existing MMFGs in terms of the `Graph Code` matrices is shown in Figure 3c, which shows an exemplary `GC_query` and an exemplary adapted version `GC_query`.

To further optimize the execution of such a query, we construct a compressed `Graph Code GC_query-C` by deleting all rows and columns with zero values from an adapted `Graph Code`. This `GC_query-C` provides an excellent basis for comparison algorithms, as it typically contains very few entries, which would also reduce the number of required matrix comparison operations. In our example, `GC_query-C` would semantically represent a search for images containing a red watch (see blue coloured fields of Figure 3c). Instead of traversing feature graphs to match sub-graphs, a `GC_query` comparison based on `Graph Codes` employs matrix-operations to find relevant `Graph Codes` based on their similarity to the `GC_query` implemented by the metric $M_{GC}$. This approach highly enables the use of Machine Learning, Pattern Matching, and specialized hardware for parallelisation of query execution, which is described now in more detail.

**Information Retrieval based on Graph Codes** utilizes the introduced retrieval function $IR_{GC}(GC_{query}) = (GC_1, ..., GC_n)$, which returns a list of `Graph Codes` ordered
by relevance implemented on basis of the similarity metric $M_{GC} = (M_F, M_{FR}, M_{RT})$ and thus directly represents the retrieval result in form of a ranked list. The calculation of this ranked list can be performed in parallel, if specialized hardware is available. For a given query Graph Code $GC_{Query}$, a similarity calculation with each Graph Code $GC$ of the collection is performed, based on the Graph Code metric $M_{GC}$. Compared to graph-based operations, matrix-based algorithms can be highly parallelized and optimized. In particular, modern GPUs are designed to perform a large number of independent calculations in parallel [38]. Thus, the comparison of two Graph Codes can be done in $O(1)$ on appropriate hardware. It is notable, that even current Smartphones or Tablets are produced with specialized hardware for parallel execution and ML tasks like Apple’s A14 bionic chip [39]. Therefore, the Graph Encoding Algorithm also performs well on Smartphones or Tablets. In the section V of this paper, we provide detailed facts and figures. The basic algorithm for this comparison and ordering is outlined in pseudocode below:

for each $GC$ in collection

--- parallelize ---
calculate the intersection matrices of $GC_{Query}$ and $GC$
--- parallelize each ---
calculate $M_F$ of $GC_{Query}$ and $GC$
calculate $M_{FR}$ of $GC_{Query}$ and $GC$
calculate $M_{RT}$ of $GC_{Query}$ and $GC$
--- end parallelize each ---
compare
--- end parallelize ---
order result list according to value of $M_F$
value of $M_{FR}$ where $M_F$ is equal
value of $M_{RT}$ where $M_F$ and $M_{FR}$ are equal
return result list

To calculate the ranked result list, this algorithm thus utilizes the three metrics $M_F$, $M_{FR}$ and $M_{RT}$ in a way, that first, the similarity according to $M_F$ (i.e. equal vocabulary terms) is calculated. For those elements, that have equal vocabulary terms, additionally the similarity value of $M_{FR}$ for similar feature relationships is applied for ordering. Also, for those elements with similar relationships (i.e. edges), we also apply the metric $M_{RT}$, which compares edge types. Hence, the final ranked result list for a $GC_{Query}$ Graph Code is produced by applying all 3 Graph Codes metrics to the collection.

**Summarizing this section,** we discussed the conceptual details, their mathematical background and formalisation, a conceptual description of the and algorithms for processing Graph Codes and their application for MMIR. We introduced Graph Code feature vocabularies and dictionaries as a foundation for further modeling. Especially Graph Code dictionary vectors are the basis for several operations and provide a clearly defined, indexed list of vocabulary terms for each Graph Code. The design of MMIR applications can employ application-wide dictionaries or dictionaries for smaller or individual vocabularies, which provides high flexibility in the application design when using Graph Codes. In this section, we also introduced an example to illustrate the corresponding matrix operations, which is also employed as a basis for the calculation of Graph Code similarity. Similarity of Graph Codes is defined by a metric $M_{GC} = (M_F, M_{FR}, M_{RT})$, which addresses different properties of the underlying MMFG (i.e. the vocabulary terms, edge relationships, and edge relationship types). With this metric-triple, a comprehensive comparison of Graph Codes can be implemented. Based on this metric, we discussed the construction of Graph Code queries, which can be completed manually, as Query by Example, or in terms of a adaptation of existing Graph Codes and will result in a query Graph Code. This query object can be compressed and will provide an excellent basis for comparison algorithms based on the metric $M_{GC}$. We also showed, that MMIR retrieval based on Graph Codes can be highly parallelized.

**IV. IMPLEMENTATION AND TESTING**

For the implementation of algorithms and concepts, we chose Java [40] and Swift [41] as programming languages. As our corresponding frameworks like the Generic Multimedia Analysis Framework [6] are already implemented in Java, we also chose Java as a programming language for the Graph Code algorithms and built components (section III). For the later evaluation, we also implemented the Graph Code Retrieval algorithm in Swift for iOS [41] to evaluate it on an iPad Pro [42], as we wanted to investigate the use of parallelisation in terms of the A14 bionic chip [39] in this device. As the implementation basically follows the algorithms and functions described in the last section, the details of the implementation including source code can be found at Github [7], where also a jupyter notebook [43] is available. In addition to this Java implementation, we also built a second implementation based on the graph-database Neo4J [4], which we can use for comparison. Therefore, the GMAF has been extended to provide an additional MMFG-export option in the Neo4J format. Thus, in both Neo4J and the Java implementation, the same MMFGs can be used for further processing. For the comparison of

![Figure 4](image.png)

Fig. 4. Initial Testing - Highresolution image and LOG.

the Neo4J and the Java implementation, we installed both the Neo4J graph database and the Java Graph Code algorithms on
a 16" Macbook Pro (2.4 GHz, 8-Core Intel Core i9 processor, 64 GB RAM, AMD Radeon Pro 5500M 4 GB Graphics card, 2 TB SSD Storage) running MacOS 11.2 Big Sur\(^5\). We also implemented the Graph Code algorithms in Swift for iOS and ran them on an iPad Pro (13", 4. generation, A14 bionic ML chip). In addition to running the Graph Code natively on the iPad, we also ran the algorithm in the iPad Simulator on the Macbook Pro. The basis for these experiments are Graph Codes, which can be generated with different Levels Of Detail (LOD)\(^5\). In our evaluation, this generation is performed by the GMAF framework\(^6\), which provides options to determine the number of recursions used for object detection. Recursions in GMAF mean, that a detected object’s bounding box is processed again and the identified sub-objects are fused into the resulting MMFG. After some recursions, the bounding boxes become too small to represent any useful detected object and the GMAF processing terminates for this object. The higher an image’s resolution, the more recursions are possible and the higher the LOD of the detected features. To illustrate the improvement in quality, when using Graph Codes and the GMAF framework, we evaluated a given high-resolution image (see Figure 4) and applied the GMAF processing with different settings for the LOD. Figure 4 shows the results of this testing. Additionally, a selection of the detected feature vocabulary terms \(FVT_{\text{MMFG}}\) for each recursion is also given in Figure 4. All these vocabulary terms shown in Figure 4 have been actually detected by the GMAF framework. No additional meta-data have been attached to the MMFGs. This testing shows, that the LOD can be increased, if the source image is of high resolution. Based on this prerequisite, we will now focus on a detailed evaluation of our prototypical proof-of-concept implementation.

V. EVALUATION

To evaluate the performance of our proof-of-concept implementation, we follow well established methods for experiments to address efficiency (i.e. runtime behavior of the MMIR application) and effectiveness (precision and recall). Experiment 1 evaluates the efficiency of the algorithms based on the number of input graphs \(n\). Experiment 2 evaluates the effectiveness of the Graph Code Algorithm based on annotations from various datasets. As discussed in the previous section, the LOD is very important for MMIR applications. However, existing datasets do not meet the requirements of a full level-of-detail processing. The Flickr30k dataset contains only low-res images, which limits number of recursions and therefore the LOG in object-detection to level 2, as then no further objects can be identified due to the low resolution of the sample images. The DIV2K dataset provides higher resolutions, and can be employed up to a LOD of level 3, but to measure full feature detection capabilities of the GMAF, an annotated dataset of high-resolution images would have to be created and maintained. Currently, such a dataset does not exist. We are considering to create and publish such a dataset in our future work. Fortunately, for proving efficiency and effectiveness of Graph Codes, the LOD achieved with current datasets, is high enough to employ existing annotated datasets for our experiments.

Goal of our Efficiency Experiment is to compare the Graph Encoding Algorithm to standard graph algorithms. Our hypothesis is, that Graph Codes perform better than graph-traversal-based algorithms. For retrieval, the calculation of similarity is very important. Thus, we compared the retrieval algorithm of Neo4J (Node similarity) to the Graph Encoding Algorithm performed on the same machine (Java implementation) and on Apples A14 Bionic in an iPad Pro. As input for the similarity calculation we used a selection of \(c\) random images of the corresponding dataset and calculated the overall number of nodes \(n\) and edges \(e\). To illustrate the correspondence between the size of the MMFG, and the runtime behavior, we performed this experiment on existing datasets with low (Flickr30K), medium (DIV2K) resolution samples, and on a high-resolution image downloaded from Adobe Stock [44]. For the low resolution evaluation with the Flickr30k dataset, we were able to produce a LOD of level 3. The results of this experiment are shown in Figure 5a. The medium resolution evaluation with the DIV2K dataset produced LODs of level 4 and 5 and shown in Figure 5b. Finally, the high-resolution evaluation generated a LOD of level 6 with results summarized in Figure 5c. This last evaluation has also been performed on an Apple iPad Pro and on a Macbook Pro (IOS Simulator).

For all experiments, we performed the standard similarity search (production quality) of Neo4J according to the Neo4J guidelines and benchmarks [4]. Before each experiment, we cleared the Neo4J database and loaded only the nodes, that are relevant for the experiment. In the Neo4J query, we adjusted the number of recursions for the graph-search to the LOD-level of the MMFG. The corresponding Neo4J-Query is also available at Github [7]. The experiment in Figure 5c shows, that the best performance is achieved with the iPad Pro application running in Apple’s Simulator application. The reason for this is, that in this case they run natively on a Apple Macbook Pro with 64GB of memory and 8-core-CPU, which is still faster than any mobile device. It is remarkable though, that native performance on the iPad Pro is still better than any other measuring (e.g. Neo4J or Java).

For the first experiment, the Graph Code algorithms outperform current graph-traversal algorithms by greater than a factor of 5 and, more importantly, grows linearly, rather than the exponential growth of graph-traversal-based algorithms. The larger the graph becomes and the more levels it contains, the greater the difference is between classic graph-traversal algorithms and Graph Code processing. These results support our hypothesis, that Graph Codes are more effective than current graph-based algorithms for MMIR. Of course, there are many options also within Neo4J to tune and optimize the database and the algorithms, but in any case, graph-traversal will have square or exponential complexity, while Graph Codes perform linearly. Additionally, also for Graph Codes several optimisations according to the MMIR application design, are imaginable and will be addressed in our future work. Another important point for Multimedia processing is, that

\(^5\)For comparison reasons, we also installed Windows 10 on this machine and double checked, that the evaluation results, described in the next section, are independent of the operating system

\(^6\)For comparison reasons, we also installed Windows 10 on this machine and double checked, that the evaluation results, described in the next section, are independent of the operating system.
Graph Codes perform well on Smartphones or Tablets as they can utilize the existing GPU hardware of these devices. Hence, the conclusion of this experiment is, that any Multimedia application can employ fast indexing and retrieval directly on the user’s device. The goal of our Effectiveness Experiment is, to calculate precision and recall of a random set of 1000 Flickr30k images. Our hypothesis is, that precision and recall values should increase due to the higher LOD. For this experiment, we did not feed any metadata into the GMAF, which would be usual. So, the results reflect the pure object detection capabilities of the GMAF-framework without any semantic enrichment. Indexing and retrieval has been completed with Graph Codes. However, as the Flickr30K dataset has been annotated manually, many different terms are employed in describing the same objects as no common ontology has been applied. Hence, we performed two sub-experiments to reflect these flaws in dataset standardisation. In the first sub-experiment (No-Synonym), only the nouns from the queries have been employed in creating a GCSearch object. This sub-experiment will deliver results for "guitar" only, where the semantic processing of the image has detected the term "guitar". The second sub-experiment (With-Synonym) also employs synonyms for the nouns when creating the GCSearch. In this case, the GCSearch will contain also synonyms in the query. So when querying "guitar", it would also contain e.g. "banjo" or "bass" in the query. Thus, these experiments also reflect the quality of standardisation within the Flickr30K dataset.

In summary, we demonstrated the validity of the modeling and implementation by showing, that the actual results with respect to efficiency and effectiveness perform better than current reference algorithms and thus that Graph Codes are highly relevant for future MMIR applications.
VI. CONCLUSION AND FUTURE WORK

In this paper we presented Graph Codes with corresponding mathematical and algorithmic methods and objectives. We discussed the mathematical model of the algorithms and the advantages of matrix calculations compared to graph-traversal operations, by ensuring that Graph Codes can be regarded semantically equivalent. We showed results from experiments which support, that Graph Codes provide a fast and easy-to-implement solution for MMIR applications, which utilizes feature graph structures. Our experiments show, that calculations in the 2D matrix space significantly outperform graph-traversal algorithms and that the implementation of Graph Codes can be satisfactorily ported to any device (e.g. tablets or smartphones). Graph Codes do not require any databases or other prerequisites and can be executed directly within any application. This facilitates installation and deployment of Graph Code based applications.

We also discovered some open remaining challenges. As we have been focusing here on images, further research with respect to the application of Graph Codes to text, audio and video assets is possible. This would significantly increase the detected LOD in the MMPGs, when features of various Multimedia types are fused into a single MMFG. Another major challenge is, that a well annotated set of sample data and deployment of Graph Code based applications.

REFERENCES


