World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:15, No:12, 2021

Graph Codes - 2D Projections of Multimedia Feature Graphs for Fast and Effective Retrieval

Stefan Wagenpfeil, Felix Engel, Paul McKevitt, Matthias Hemmje

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.

Stefan Wagenpfeil, Felix Engel, and Matthias Hemmje are with the Faculty for Mathematics and Computer Science, University of Hagen, Germany (phone: +49/160/4071976, e-mail: stefan.wagenpfeil@fernuni-hagen.de)
Paul McKevitt is with the Academy for International Science & Research (AISR), Derry/Londonderry, Northern Ireland

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

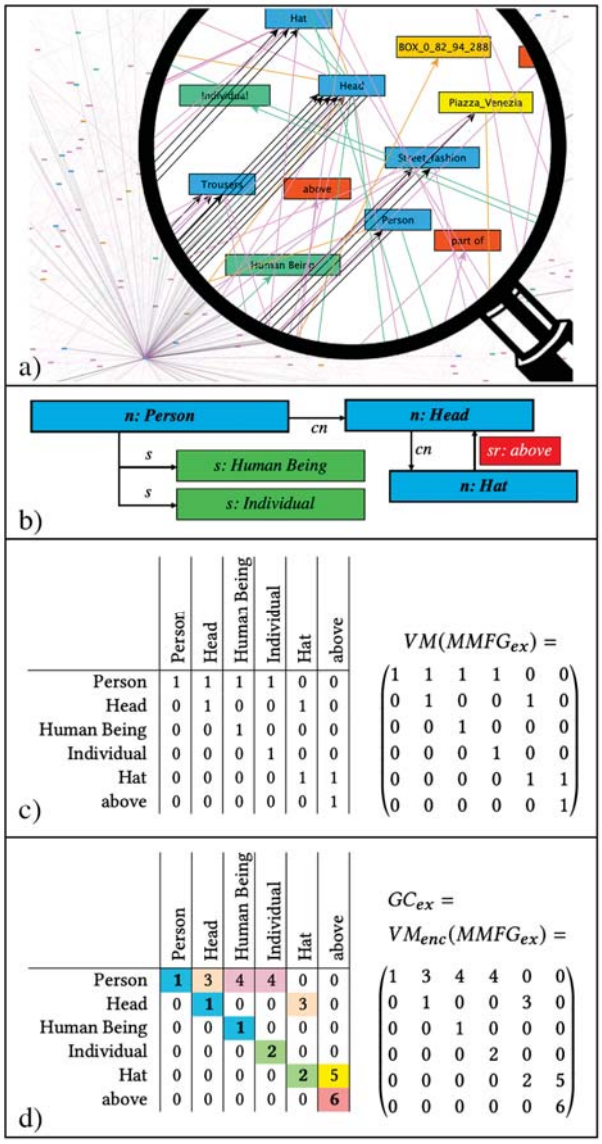


Fig. 1. Exemplary $MMFG_{ex}$ and its representations. Figure 1a shows a typical MMFG visualized as GraphML with yEd, 1b illustrates a small exemplary instance graph. 1c shows a table representation and the Valuation Matrix VM, 1d the corresponding *Graph Code*.

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 $O(n+e)^2$ (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 a 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[1]. If we apply such a f_{enc} to the above example[2], the encoded Valuation Matrix VM_{enc}, i.e. the corresponding *Graph Code* GC is shown in Figure 1d. Going beyond this encoding, later, attributes, weights, or 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 IAPTRC12 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 of 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.

[1]The 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.

[2]The 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, spacialrelationship-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 $MMFVG_{Query}$ 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 $MMFVG_{Query}$ and each element of the set of existing $MMFGs$. 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 $IR_{MMFG}(MMFG_{Query}) = \{MMFG_1, ..., MMFG_n\}$. In the case of *Graph Codes* and the corresponding algorithms, each MMFG is represented by its GC_{MMFG} and the retrieval function is $IR_{GC}(GC_{Query}) = (GC_1, ..., GC_n)$. The result of IR_{GC} is a ordered vector of all *Graph Codes* of the collection, in which $\forall GC_i \in IR_{GC} : GC_i > GC_{(i+1)}$. 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 $FVT_{MMFG} = \{ft_1, ..., ft_n\}$. 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[3]. In the *Graph Code* matrix representation, each node field (in the diagonal) of a *Graph Code* can now be unambigously mapped to an entry of its *Graph Code Dictionary* vector, which can be represented as $dict_{GC} = (ft_1, ..., ft_n)$. 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 $dict_{ex} = (Person, Head, HumanBeing, Individual, Hat, above)$ 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*

[3]Any ordering strategy can be applied, e.g. a breadth-first-search based on the MMFG structure. In the following examples, we chose a manual ordering to maximize illustration.

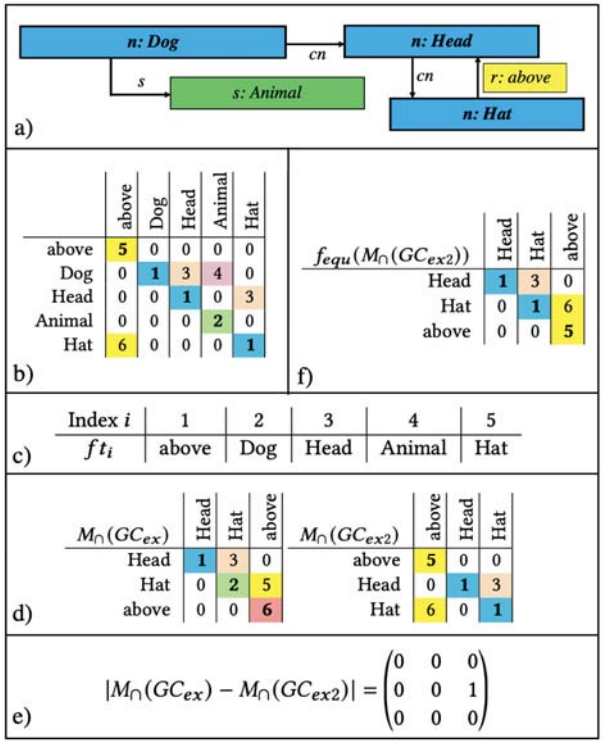


Fig. 2. $MMFG_{ex2}$ and representations. Figure 2a shows a second exemplary MMFG, 2b the corresponding *Graph Code* GC_{ex2}. Figure 2c illustrates the dictionary of GC_{ex2}, and 2d the intersections $M_\cap$. Figure 2e illustrates the subtraction, 2f the encoded intersection *Graph Code*.

Code has its own, individual dictionary-vector $dict_{GC}$, and another *Graph Code* will have a different dictionary-vector according to the content of its represented MMFG, typically $dict_{GC1} \neq dict_{GC2}$. Feature-equivalent node fields of *Graph Codes* can be determined through their corresponding *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 *Graph Codes* is relevant. Thus, the set of intersecting feature vocabulary terms $FVT_{\cap 1,2}$ of e.g. 2 MMFGs can be defined as $FVT_{\cap 1,2} = \{ft_1, ..., ft_n\} = V_{MMFVG_1} \cap V_{MMFVG_2}$. The methodology of intersecting sets can be also applied to *Graph Code* dictionaries. The intersection of two vectors $dict_\cap 1, 2$ can be defined correspondingly as $dict_{\cap 1,2} = dict_{GC_1} \cap dict_{GC_2}$. To illustrate the calculation of $dict_\cap$, we introduce a second exemplary *Graph Code* GC_{ex2} based on a $MMFG_{ex2}$, which is shown in Figure 2a.

The set FVT_{ex2} in this case is {*above, Dog, Head, Animal, Hat*} and the set $FTV_{\cap 1,2}$ of intersecting feature vocabulary terms is {*above, Head, Hat*}. The dictionary-vector $dict_{ex2}$ thus is $dict_{ex2}$ = *(above, Dog, Head, Animal, Hat)*. Figure 2b shows its table representation and Figure 2c its list representation $dict_{ex2}$. The vector $dict_{\cap ex1,2}$ represents the dictionary of intersecting vocabulary terms and only contains the subset of vocabulary terms of $dict_{ex}$, where a equal vocabulary term can be found in $dict_{ex2}$. The order of intersecting vocabulary terms in $dict_{\cap 1,2}$ is given by the order of $dict_{ex}$. From an algorithmic perspective, this means, that all elements of $dict_{ex}$ are deleted, that cannot be found in $dict_{ex2}$. The index position of $dict_{\cap 1,2}$ typically is different from both $dict_1$ and $dict_2$. Based on these dictionary-vectors, a translation of equivalent *Graph Code* positions can be performed, as each feature vocabulary term has a unique position within each of the *Graph Code's* dictionaries.

Applications will typically utilize a collection of MMFGs and their corresponding *Graph Codes*. The overall feature term vocabulary $FVT_{Coll} = \{ft_1, ..., ft_c\}$ containing c vocabulary terms of such a collection of n MMFGs can be defined as the union of all MMFG's feature term vocabularies and also be represented by the union of all *Graph Code Dictionaries* $dict_\cup$: $FVT_{Coll} = \bigcup_{i=1}^{n} FVT_{MMFG_i}$, where $\forall i, j < n : dict_\cup = dict_i \times dict_j$. In this $dict_\cup$ dictionary-union-vector, the $\times$-operation for calculating the union of dictionary-vectors is implemented by traversing all the collection's $dict_i$ dictionaries and collecting unique dictionary vocabulary terms into a single dictionary-vector. In our example with $dict_{ex}$ and $dict_{ex2}$, the calculated $dict_\cup$ = *(Person, Head, Human Being, Individual, Hat, above, Dog, Animal)*. If a $dict_\cup$ is calculated for the complete collection of *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 *Graph Codes* having similar sizes, but different vocabulary terms, leading to an increase of V_{Coll}. The Oxford English Dictionary [34] e.g., contains 170.000 english words[4] 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 170.000 $\times$ 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:

Application-wide dictionary: in this scenario, we assume that any *Graph Code* will be processed with the dictionary-vector terms $dict_\cup$. 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 *Graph Codes* have to be adjusted whenever new indexing terms are detected (the size of $dict_\cup$ increases) or whenever existing Multimedia feature content is removed from the collection (the size of $dict_\cup$ decreases). The big advantage of this approach is, that all *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 *Graph Codes* can be very expensive. In this case, the following scenario should be preferred.

[4]Translation and multilingual support is not in scope of this paper and does not affect the general concept of *Graph Codes*.

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 $dict_{\cap}$. 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 $dict_{\cup}$), this approach can be very efficient, when *Graph Codes* vary significantly within a collection.

Applied to GC_{ex} and GC_{ex2} of our example above, the application-wide dictionary $dict_{\cup}$ would give *Graph Codes* with a size of 9×9 matrix fields, whereas $dict_{\cap}$ would give a intersection matrix of 3×3 fields. This intersection matrix $M_{\cap}(GC)$ can be calculated from a GC by removing any rows and columns, that are not part of $dict_{\cap}$. Figure 2d shows the intersection matrices of GC_{ex} and GC_{ex2}.

For comparison of these intersection matrices, we wish to apply the standard matrix subtraction. However, due to the different orders of $dict_{ex}$ and $dict_{ex2}$, the matrix field positions of the matrices do not represent the same feature vocabulary terms. For example, the field (2,1) of GC_{ex} represents the relationship between *Hat* and *Head*, but the equivalent relationship in GC_{ex2} is located in field (3,2). To solve this, we introduce a equivalence function $f_{equ}(M_{\cap})$, 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 $dict_{\cup}$.

Thus, equivalence of a matrix field $(m_{i,j})$ in $M_{\cap}(GC_i)$ and a matrix field $(n_{k,l})$ in $M_{\cap}(GC_j)$ and corresponding dictionary vectors $dict_i$ and $dict_j$ can be defined as: $\forall(m_{i,j}) \in M_{\cap}(GC_i), \forall(n_{k,l}) \in M_{\cap}(GC_j)$:

$$M_{\cap}(GC_i) \sim f_{equ}(M_{\cap}(GC_j)) \Leftrightarrow$$

$$dict_i(i) = f_{equ}(dict_j(k)) \wedge dict_i(j) = f_{equ}(dict_j(l))$$

In the case of comparing only 2 *Graph Codes*, $dict_{\cup}$ 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, $dict_{ex2} = (above, Head, Hat)$ would be re-ordered to match $dict_{ex} = (Head, Hat, above)$. 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.

a)

$AM(M_{\cap}(GC_{ex}))$	Head	Hat	above
Head	1	1	0
Hat	0	1	1
above	0	0	1

$AM(f_{equ}(M_{\cap}(GC_{ex2})))$	Head	Hat	above
Head	1	1	0
Hat	0	1	1
above	0	0	1

b)

$M_{\cap}(GC_{ex})$	Head	Hat	above
Head	1	3	0
Hat	0	2	5
above	0	0	6

$f_{equ}(M_{\cap}(GC_{ex2}))$	Head	Hat	above
Head	1	3	0
Hat	0	1	6
above	0	0	5

c)

GC_{Query}	Person	Jim	Watch	red
Person	1	3	5	1
Jim	0	2	0	0
Watch	0	0	1	1
red	0	0	0	1

GC'_{Query}	Person	Jim	Watch	red
Person	0	0	0	0
Jim	0	0	0	0
Watch	0	0	1	1
red	0	0	0	1

Fig. 3. Examples of *Graph Code* metrics. Figure 3a shows the Adjacency Matrices of the equivalent *Graph Code* fields of GC_{ex} and GC_{ex2}. 3b shows the intersections $M_{\cap}$, and Figure 3c the GC_{Query} of our example.

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 $M_{GC} = (M_F, M_{FR}, M_{RT})$ as a triple of metrics containing a *feature-metric* M_F, a *feature-relationship-metric* M_{FR} and a *feature-relationship-type-metric* M_{RT}.

The Graph Code Feature Metric M_F can be employed to calculate the similarity of *Graph Codes* according to the intersecting set of dictionary vocabulary terms. M_F is defined as the ratio between the cardinality of $dict_{\cap}$ the intersecting dictionary vocabulary terms and the cardinality $dict_i$ 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: $M_F(GC_i, GC_j) = \frac{|dict_{\cap}|}{|dict_i|}$. Thus, the more features are common in e.g., 2 MMFGs, the higher the similarity value based on M_F - independent of the relationships between these corresponding MMIR features. In the above example, the numerical distance between GC_{ex} and GC_{ex2} based on the metric M_F is $M_F(GC_{ex}, GC_{ex2}) = \frac{|dict_{\cap ex1,2}|}{|dict_{ex}|} = \frac{3}{6} = 0.5$.

The Graph Code Feature Relationship Metric M_{FR} 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 $M_\cap$ of two *Graph Codes*. We base this metric on the non-diagonal fields of the Adjacency Matrix $AM(M_\cap)$ (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. $M_{FR}(GC_i, GC_j) = \frac{\sum AM(M_{\cap i,j})-n}{|AM(M_{\cap i})|-n}$. Thus, M_{FR} represents the ratio between the number of non-zero edge-representing matrix fields and the overall number of equivalent and intersecting edge-representing matrix fields of e.g. two *Graph Codes*. In this way, the metric M_{FR} counts all edges existing between source and target nodes, independent of the equivalence of the edges' types.

The $AM(M_\cap(GC_ex))$ and the equivalent matrix $f_{equ}(M_\cap(GC_{ex2}))$ of our example is shown in Figure 3a. Looking at the two *Graph Codes* in Figure 3a, there are six potential positions representing edges: 3 fields in the upper right of the diagonal and 3 fields in the lower left. Out of these possible positions, only 2 contain edges. These are located in matrix positions (2,1) and (3,2). Thus, only 2 out of 6 possible edge representing matrix field positions have a non-zero value. Thus, the numerical distance of the metric $M_{FR}(GC_{ex}, GC_{ex2}) = \frac{\sum AM(M_{\cap i,j})-n}{|AM(M_{\cap i})-n|} = \frac{5-3}{9-3} = \frac{2}{6} = 0.33$. Note, that currently only the existence of an edge - independent from its type - is employed for the metric M_{FR}. However, also the type of each relationship can indicate additional similarity. Therefore, we will introduce an edge-type-based metric in the next section.

The Graph Code Feature Relationship Type Metric M_{RT} is based on the feature-relationship-types of *Graph Codes*. As the *Graph Code* encoding function f_{enc} encodes different MMFG edge-types with different base values, feature-relationship-type-similarity can only exist, when the edge-types represented by equivalent matrix fields of *Graph Codes* are equal. In case of M_{RT}, calculations are performed no longer on the adjacency matrices of *Graph Codes*, but on the $M_\cap$ matrices of the *Graph Codes* as shown in Figure 3b. A matrix field is equal to another, if the subtraction of their values returns zero. If all equivalent fields are equal, the sum of these fields is zero. While M_{FR} is based on the pure existence of a non-zero edge representing matrix field, M_{RT} additionally employs the value of this matrix field (representing the relationship type) and therefore represents the ratio between the sum of all non-diagonal matrix fields and their cardinality.

$$M_{RT}(GC_i, GC_j) = \frac{\sum_{i,j}^{n,i\neq j}(|M_{\cap i} - M_{\cap j}|)}{|M_{\cap i}| - n}$$

. In our example, the difference of these two intersection matrices for non-diagonal fields (i.e. $i \neq j$) is shown in Figure 2e. Thus, the mathematical sum of this matrix is 1. This means, that 1 out of 6 possible fields had a different edge type value. The numerical distance of the metric M_{RT} for these two *Graph Codes* can be calculated as

$$M_{RT}(GC_i, GC_j) = \frac{\sum_{i,j}^{n,i\neq j}(|M_{\cap i} - M_{\cap j}|)}{|M_{\cap i}| - n} = \frac{1}{9-3} = \frac{1}{6} = 0.16$$

.

Thus, in terms of our example, the overall similarity M_{GC} between GC_{ex} and GC_{ex2} is $M_{GC}(GC_{ex}, GC_{ex2}) = (M_F, M_{FR}, M_{RT}) = (0.5, 0.33, 0.16)$. This means, that the similarity based on common vocabulary terms M_F is 0.5, the similarity based on common edge positions M_{FR} 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 $MMFG_{Query}$ 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 $MMFG_{Query}$ 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 $IR_{GC}(GC_{Query}) = (GC1, ..., GCn)$, 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 Code* 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

LOD	# n	# e	selected FVT_{MMFG} example terms
0	53	204	Person, Travel, Piazza Venezia, Joy
1	71	274	Pants, Top, Pocket, Street fashion
2	119	475	Camera, Bermuda Shorts, Shoe
3	192	789	Sun Hat, Fashion accessory, Watch, Bergen County (Logo)
4	228	956	Mouth, Lip, Eyebrow, Pocket, Maroon
5	274	1189	Leather, Grey, Single-lens reflex camera

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_{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

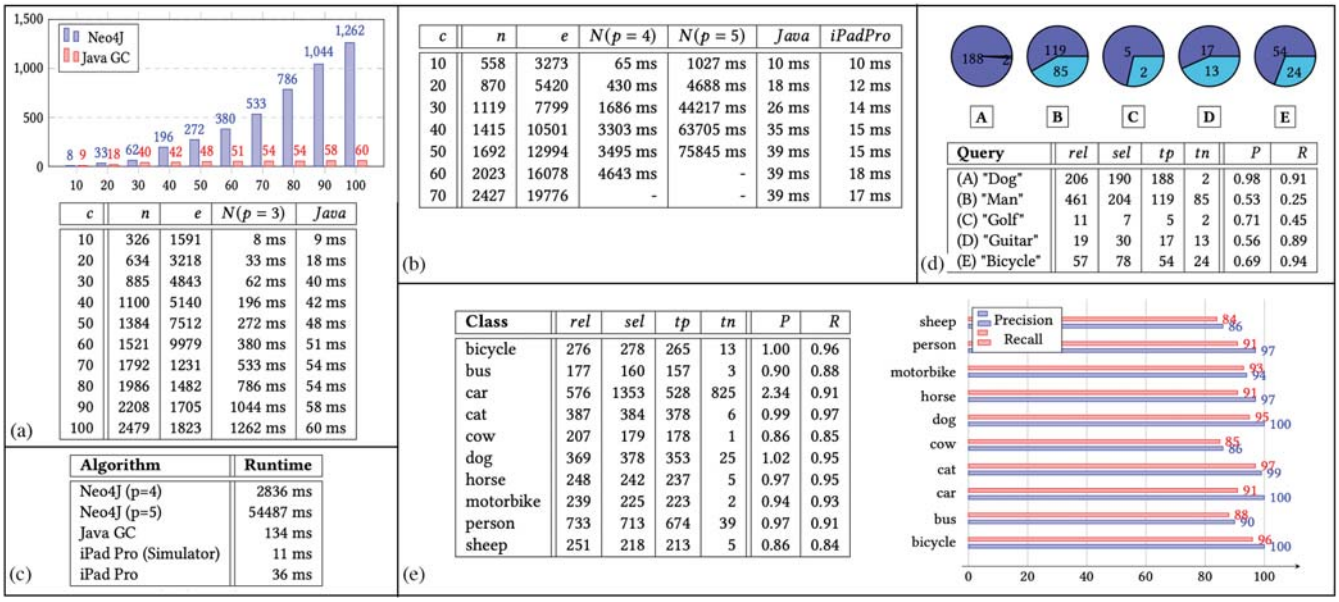
To evaluate the performance of our proof-of-concept implementation, we follow well established methods for experiments to address efficiency (i.e. runtime behavior of the MMIR application) and effectiveness (precision and recall). Experiment 1 evaluates the efficiency of the algorithms based on the number 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



c	n	e	N(p = 3)	Java
10	326	1591	8 ms	9 ms
20	634	3218	33 ms	18 ms
30	885	4843	62 ms	40 ms
40	1100	5140	196 ms	42 ms
50	1384	7512	272 ms	48 ms
60	1521	9979	380 ms	51 ms
70	1792	1231	533 ms	54 ms
80	1986	1482	786 ms	54 ms
90	2208	1705	1044 ms	58 ms
100	2479	1823	1262 ms	60 ms

c	n	e	N(p = 4)	N(p = 5)	Java	iPadPro
10	558	3273	65 ms	1027 ms	10 ms	10 ms
20	870	5420	430 ms	4688 ms	18 ms	12 ms
30	1119	7799	1686 ms	44217 ms	26 ms	14 ms
40	1415	10501	3303 ms	63705 ms	35 ms	15 ms
50	1692	12994	3495 ms	75845 ms	39 ms	15 ms
60	2023	16078	4643 ms	-	39 ms	18 ms
70	2427	19776	-	-	39 ms	17 ms

Algorithm	Runtime
Neo4J (p=4)	2836 ms
Neo4J (p=5)	54487 ms
Java GC	134 ms
iPad Pro (Simulator)	11 ms
iPad Pro	36 ms

Query	rel	sel	tp	tn	P	R
(A) "Dog"	206	190	188	2	0.98	0.91
(B) "Man"	461	204	119	85	0.53	0.25
(C) "Golf"	11	7	5	2	0.71	0.45
(D) "Guitar"	19	30	17	13	0.56	0.89
(E) "Bicycle"	57	78	54	24	0.69	0.94

Class	rel	sel	tp	tn	P	R
bicycle	276	278	265	13	1.00	0.96
bus	177	160	157	3	0.90	0.88
car	576	1353	528	825	2.34	0.91
cat	387	384	378	6	0.99	0.97
cow	207	179	178	1	0.86	0.85
dog	369	378	353	25	1.02	0.95
horse	248	242	237	5	0.97	0.95
motorbike	239	225	223	2	0.94	0.93
person	733	713	674	39	0.97	0.91
sheep	251	218	213	5	0.86	0.84

Fig. 5. *Graph Code* experiment results. 5a shows the efficiency of the Java *Graph Code* algorithms compared to Neo4J on the Flickr30k dataset, 5b the results of the DIV2K dataset. Figure 5c provides a runtime comparison, 5d and 5e detailed results of the Precision and Recall evaluations.

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 GC_{Search} 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 GC_{Search}. In this case, the GC_{Search} 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. Figure 5d shows values for the relevant objects rel in the dataset, the selection sel by the algorithm, the number of true positive results tp, the number of true negative results tn, precision P and recall R. We investigated further these values of P and R and discovered some flaws in the Flickr30k dataset. In our manual review of the annotations and corresponding images, we found many inaccurate annotations. About 80% of the "false negatives" i.e. actually correct, but with incorrect input image annotations. In general, the Flickr30k dataset would have to be reviewed and corrected in order to perform further tests. After this discovery, an additional experiment with the PASCAL VOC dataset [33] has been performed. This dataset comes with pre-defined object classes and annotations. An experiment of R. Scherer [45], published in 2020 and proposing a "Salient Object Detector and Descriptor by Edge Crawler" algorithm produced an average precision of 78.58%. Experiment results are shown in Figure 5e and demonstrate, that the GMAF and *Graph Code* processing increased the average precision to 96.26%.

Discussin this experiment, the object detection of GMAF and the corresponding *Graph Code* representation actually is more accurate than the annotation file metadata of the Flickr30k dataset. In "real world" tests, we were able to produce Precision and Recall results of ¿ 97%. The experiment with the PASCAL dataset also supports this and provided about 15% better results with an average precision of 96% and a recall of 92%. These results support our hypothesis, that the *Graph Codes* significantly increase the effectiveness of MMIR due to their higher LOD. Thus, the accuracy of the results in MMIR applications is even higher than any manual annotation as the GMAF fuses detected objects with textual annotations found in Social Media posts, Metadata, or on corresponding websites.

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 MMFGs, when features of various Multimedia types are fused into a single MMFG. Another major challenge is, that a well annotated set of sample data for high-resolution, feature-rich MMIR applications has to be created. Our future work will address these challenges to further show, that *Graph Codes* provide a highly efficient extension to Multimedia Databases for indexing and retrieval.

References

[1] E. Spyrou, D. Iakovidis, and P. Mylonas, *Semantic Multimedia Analysis and Processing*. Boca Raton, Fla: CRC Press, 2017, ISBN: 978-1-351-83183-3.

[2] C. J., "Social media - statistics & facts", Statista Inc., https://www.statista.com/topics/1164/social-networks/, Tech. Rep., Aug. 2020.

[3] W3C.org, "W3c semantic web activity", W3C.org, http://w3.org/2001/sw, Tech. Rep., Jul. 2020.

[4] N. Inc. (2020). "Neo4j", [Online]. Available: https : / / neo4j . com, Download: 01.10.2020.

[5] S. Wagenpfeil, F. Engel, P. M. Kevitt, and M. Hemmje, "Ai-based semantic multimedia indexing and retrieval for social media on smartphones", *Information*, vol. 12, no. 1, 2021, ISSN: 2078-2489. DOI: 10.3390/info12010043. [Online]. Available: https://www.mdpi.com/2078-2489/12/1/43.

[6] S. Wagenpfeil, "Gmaf prototype", University of Hagen, Faculty of Mathematics and Computer Science, http://diss.step2e.de:8080/GMAFWeb/, Tech. Rep., Jul. 2020.

[7] ——, (Sep. 2021). "Github repository of gmaf and mmfvg", [Online]. Available: https : / / github . com / stefanwagenpfeil / GMAF/, Download: 03.10.2020.

[8] S. Wagenpfeil and M. Hemmje, *Towards ai-bases semantic multimedia indexing and retrieval for social media on smartphones*, SMAP 2020 Conference, Sep. 2020.

[9] J. Beyerer, M. Richter, and M. Nagel, *Pattern Recognition - Introduction, Features, Classifiers and Principles*. Berlin: Walter de Gruyter GmbH & Co KG, 2017, ISBN: 978-3-110-53794-9.

[10] O. Kurland and J. S. Culpepper, "Fusion in information retrieval: Sigir 2018 half-day tutorial", in *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '18, New York, NY, USA: Association for Computing Machinery, 2018, 1383–1386, ISBN: 9781450356572. DOI: 10.1145/3209978.3210186. [Online]. Available: https://doi.org/10.1145/3209978.3210186.

[11] J. Leveling, "Interpretation of coordinations, compound generation, and result fusion for query variants", in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '13, Dublin, Ireland: Association for Computing Machinery, 2013, 805–808, ISBN: 9781450320344. DOI: 10.1145/2484028.2484115. [Online]. Available: https://doi.org/10.1145/2484028.2484115.

[12] A. Bhute, B. Meshram, and H. Bhute, "Multimedia indexing and retrieval techniques: A review", *International Journal of Computer Applications*, vol. 58, pp. 35–42, Nov. 2012. DOI: 10.5120/9264-3443.

[13] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges", *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 2, no. 1, 1–19, Feb. 2006, ISSN: 1551-6857. DOI: 10.1145/1126004.1126005. [Online]. Available: https://doi.org/10.1145/1126004.1126005.

[14] C. Hernández-Gracidas, A. Juárez, L. E. Sucar, M. Montes-y Gómez, and L. Villaseñor, "Data fusion and label weighting for image retrieval based on spatio-conceptual information", in *Adaptivity, Personalization and Fusion of Heterogeneous Information*, ser. RIAO '10, Paris, France: Le Centre des Hautes etudes Internationales, 2010, 76–79.

[15] R. Dufour, Y. Estève, P. Deléglise, and F. Bechet, "Local and global models for spontaneous speech segment detection and characterization", Jan. 2010, pp. 558 –561. DOI: 10.1109/ASRU.2009.5372928.

[16] V. S. Subrahmanian, *Principles of Multimedia Database Systems*. San Francisco: Morgan Kaufmann Publishers, 1998, ISBN: 978-1-558-60466-7.

[17] Shih-Fu Chang, T. Sikora, and A. Purl, "Overview of the mpeg-7 standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 688–695, 2001.

[18] FFMpeg.org, "Ffmpeg documentation", FFMpeg.org, http://ffmpeg.org, Tech. Rep., Jul. 2020.

[19] X. Mu, "Content-based video retrieval: Does video's semantic visual feature matter?", in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '06, Seattle, Washington, USA: Association for Computing Machinery, 2006, 679–680, ISBN: 1595933697. DOI: 10.1145/1148170.1148314. [Online]. Available: https://doi.org/10.1145/1148170.1148314.

[20] F. Gurski, D. Komander, and C. Rehs, "On characterizations for subclasses of directed co-graphs", *CoRR*, vol. abs/1907.00801, 2019. arXiv: 1907.00801. [Online]. Available: http://arxiv.org/abs/1907.00801.

[21] yWorks GmbH, "Yed graph editor", yWorks GmbH, https://www.yworks.com/products/yed, Tech. Rep., Aug. 2020.

[22] Sciencedirect.com. (2020). "Adjacency matrix", [Online]. Available: https://www.sciencedirect.com/topics/mathematics/adjacency-matrix.

[23] G. Fischer, *Lineare Algebra*. Springer Spektrum, 2014, ISBN: 978-3-658-03945-5.

[24] Y. Yuan, G. Wang, L. Chen, and H. Wang, "Graph similarity search on large uncertain graph databases", *The VLDB Journal*, vol. 24, no. 2, pp. 271–296, Apr. 2015, ISSN: 0949-877X. DOI: 10.1007/s00778-014-0373-y. [Online]. Available: https://doi.org/10.1007/s00778-014-0373-y.

[25] M. Needham, *Graph Algorithms*. 1005 Gravenstein Highway North Sebastopol CA 95472: O'Reilly Media Inc., 2019, ISBN: 978-1-492-05781-9.

[26] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec", *CoRR*, vol. abs/1710.02971, 2017. arXiv: 1710.02971. [Online]. Available: http://arxiv.org/abs/1710.02971.

[27] T. R. Conference. (Jan. 2020). "Datasets", [Online]. Available: https://trec.nist.gov/data.html.

[28] T. D. Science. (Nov. 2018). "Over 1.5 tb's of labeled audio datasets", [Online]. Available: https://towardsdatascience.com/a-data-lakes-worth-of-audio-datasets-b45b88cd4ad.

[29] Google.com. (Jan. 2021). "A large and diverse labeled video dataset for video understanding research", [Online]. Available: http://research.google.com/youtube8m/.

[30] P. Young. (2016). "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions", [Online]. Available: http://shannon.cs.illinois.edu/DenotationGraph/.

[31] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017. [Online]. Available: http : / / www . vision . ee . ethz . ch / ~timofter / publications/Agustsson-CVPRW-2017.pdf.

[32] M. Grubinger, P. Clough, H. Müller, and T. Deselaers, "The iapr tc12 benchmark: A new evaluation resource for visual information systems", *Workshop Ontoimage*, Oct. 2006.
[33] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge", *International Journal of Computer Vision*, vol. 88(2), pp. 303–338, 2010.
[34] (Aug. 2021). "The oxford english dictionary", Oxford University Press, [Online]. Available: https://www.oed.com, Download: 11.08.2021.
[35] W3C.org, "Sparql query language for rdf", W3C.org, https://www.w3.org/TR/sparql11-overview/, Tech. Rep., Aug. 2013.
[36] H. Jung and W. Kim, "Automated conversion from natural language query to sparql query", *Journal of Intelligent Information Systems*, 2020, ISSN: 1573-7675. DOI: 10.1007/s10844-019-00589-2. [Online]. Available: https://doi.org/10.1007/s10844-019-00589-2.
[37] I. Schmitt, N. Schulz, and T. Herstel, "Ws-qbe: A qbe-like query language for complex multimedia queries", in *11th International Multimedia Modelling Conference*, 2005, pp. 222–229.
[38] Nvidia.com. (Nov. 2020). "Rtx 2080", [Online]. Available: https://www.nvidia.com/de-de/geforce/graphics-cards/rtx-2080/.
[39] Apple.com. (Oct. 2020). "Apple iphone 12", [Online]. Available: https://www.apple.com/iphone/, Download: 13.10.2020.
[40] Oracle.com, "Java enterprise edition", Oracle.com, https://www.oracle.com/de/java/technologies/java-ee-glance.html, Tech. Rep., Aug. 2020.
[41] Apple.com. (Nov. 2020). "Apple development programme", [Online]. Available: http://developer.apple.com.
[42] Apple. (Oct. 2020). "Apple ipad pro", [Online]. Available: https://www.apple.com/ipad-pro/, Download: 13.10.2020.
[43] Jupyter.org. (Oct. 2020). "The jupyter notebook", [Online]. Available: https://jupyter.org, Download: 13.10.2020.
[44] Adobe.com. (Oct. 2020). "Adobe stock", [Online]. Available: https://stock.adobe.com, Download: 02.10.2020.
[45] R. Scherer, *Computer Vision Methods for Fast Image Classification and Retrieval*. Gewerbestrase 11, 6330 Cham, Switzerland: Springer Nature Switzerland AG, 2020, pp. 53–55, ISBN: 978-3-030-12194-5.