MULTI-STAGE QUERIES AND TEMPORAL SCORING IN VITRIVR

Silvan Heller[1], *Loris Sauter*[1], *Heiko Schuldt*[1], *and Luca Rossetto*[2]

[1]Department of Mathematics and Computer Science
University of Basel, Basel, Switzerland
[2]Department of Informatics
University of Zurich, Zurich, Switzerland

ABSTRACT

The increase in multimedia data brings many challenges for retrieval systems, not only in terms of storage and processing requirements but also with respect to query formulation and retrieval models. Querying approaches which work well up to a certain size of a multimedia collection might start to decrease in performance when applied to larger volumes of data. In this paper, we present two extensions made to the retrieval model of the open-source content-based multimedia retrieval stack vitrivr which enable a user to formulate more precise queries which can be evaluated in a staged manner, thereby improving the result quality without sacrificing the system's overall flexibility. Our retrieval model has shown its scalability on V3C1, a video collection encompassing approx. 1000 hours of video.

***Index Terms*—** Staged Query, Temporal Query, Query execution model

1. INTRODUCTION

The continuous production and the resulting steady growth of multimedia data continues to pose challenges for multimedia management and retrieval systems. Some of these challenges are technical in nature, coming from the increased need of storage and processing capabilities. Others, in turn, arise from certain system interaction patterns and query modes decreasing their effectiveness with the increase in data that must be handled. The vitrivr retrieval stack has in the past successfully used a multi-step score-based late-fusion scheme on a series of independent features in order to perform content-based retrieval with a flexible and user-configurable notion of similarity. By independently retrieving top-k results per query modality (e.g. tag, sketch) and feature category (e.g. edges, color), some results are returned almost instantly with the quality of results improving as more features return their results. While this system works well up to certain collection sizes, it begins to show problems as the data size increases. These challenges are already noticeable at the scale of V3C [1], which is small in comparison to larger state-of-the-art datasets such as YFC100M [2]. One of those challenges is that because retrieval per feature runs parallel and vitrivr uses 20+ features, k would have to equal the number of vectors to get the ”correct” score for all results. vitrivr also has been focused on the retrieval of one element from the collection without considering its temporal context, despite this context being relevant in video retrieval. In order to address this, we introduce two extensions to the vitrivr retrieval model, namely multi-staged queries and temporal scoring. With multi-staged queries, we add explicit filtering with different modalities to the query formulation, for example by first searching for tags and then ranking the results by their similarity to a sketch. For temporal scoring, we use a context-aware algorithm to tackle the existing problem of rank aggregation by trying to find result groups which match the temporal query order.

The remainder of this paper is structured as follows: Section 2 outlines the retrieval model used by vitrivr. Staged queries are introduced in Section 3 and Section 4 gives an overview of temporal scoring in vitrivr. Section 5 concludes.

2. THE VITRIVR RETRIEVAL MODEL

The vitrivr [3] stack consists of three core components, a dedicated multimedia retrieval database *Cottontail DB*, the retrieval engine *Cineast* [4] and a web-based user interface *vitrivr-ng* [5]. The entire stack is available on Github.[1]

vitrivr offers content-based retrieval functionality with multiple query modalities and similarity notions. In order to achieve this, the query processing component of the vitrivr stack — Cineast — provides a runtime for an arbitrary number of independent *feature modules* which can be selectively used depending on the properties of the queries. While the feature modules are free to perform any retrieval operation, as long as they produce a scored list of the most similar items, most of them perform vector space retrieval in a form analogous to what is outlined in Listing 1.

[1]`https://github.com/vitrivr/`

978-1-7281-1485-9/20/$31.00 ©2020 IEEE

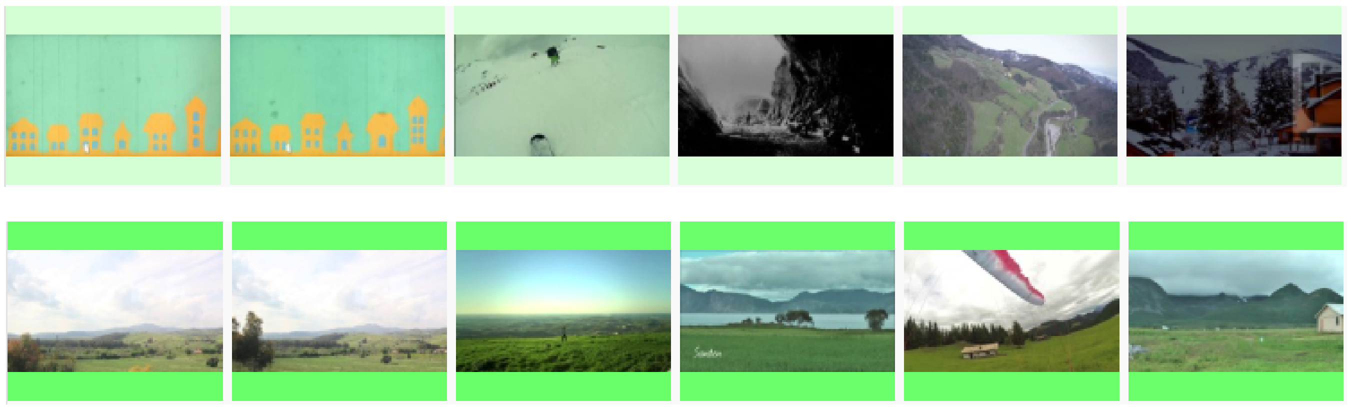

Fig. 1: Results for both a non-staged query (top) based on a sketch and tag and a multi-staged query (bottom) filtering first on tags (‘mountain’ and ‘meadow’) and then on a color sketch. While the top line contains undesirable results which are either only a match for tags or sketch, the bottom line only contains matches for both.

```
SELECT id,
  score(
    feature_transform(query),
    feature_vectors
  ) AS score
FROM feature_table
LIMIT 1000
ORDER BY score DESC
```

List. 1: Pseudo-SQL for per-feature vector space retrieval, retrieving the top 1000 relevant results.

vitrivr’s feature modules receive the complete query representation as it was specified through Cineast’s API and independently perform similarity search based on the components of the query which are relevant for them. Each query can consist of several *terms*, each describing the content of a different modality, such as visual, auditory, textual, semantic, etc. The individual retrieval results produced independently by the feature modules are aggregated using a two-step score-based weighted fusion scheme. The first step is performed by Cineast using groups of feature modules with similar notions of similarity, such as colors, edges, local points of interest, semantic annotations, etc. The composition of, as well as the weights within these groups of features, referred to as *categories* is configurable at the startup of Cineast. The second fusion step is performed by the user interface to generate the final scored list of retrieved results. The weights for this fusion are configurable by the user and can be adjusted in real time. The only requirement imposed on the feature modules for this fusion process to work is the consistency of the *ids* assigned to the described items across all modules. This is ensured during the *feature extraction* process where the ids are assigned by the Cineast extraction runtime rather than the individual feature modules. A described item in this case is either the entire document (in terms of images and 3d models) or a temporal segment of a document (in case of video or audio files).

Despite the independence of the retrieval processes performed by the feature modules, the fusion process causes results which are scored by multiple modules to bubble to the top of the final result list. Since elements which are not present in any individual result list of a module are implicitly scored with 0, modules can also act as a sort of filter by returning a list of all *relevant* elements scored with the highest possible score 1. This will cause all elements which do not meet the criterion to be lower in the list than those which do, provided they received the same score from the other modules. For example, if a user wishes to retrieve all scenes of a mountainous terrain and blue sky with a green meadow in the foreground, this query could be specified using a rough color sketch outlining the color distribution combined with a tag-based query on ‘mountain’ and ‘meadow’. The feature modules responsible for color would then retrieve all elements with a matching color distribution while the modules concerned with semantic tags would return all elements with matching tags. The elements in the intersection of these two categories would have a higher score than the elements only present in either list.

This process, however, relies on the following two assumptions. First, the number of relevant results with respect to the filter is sufficiently small, as to not be affected by the maximum number of results considered by every module. Second, the query is sufficiently selective so that the same elements which are relevant for the filter also appear in the result list of the other modules. For any sufficiently large collection of items, however, there are queries for which these assumptions do not hold. For the previous example, this could mean that the top-n results with respect to the color distribution do not contain any mountains, leaving the user with a list of results which match well to one aspect of the query with-

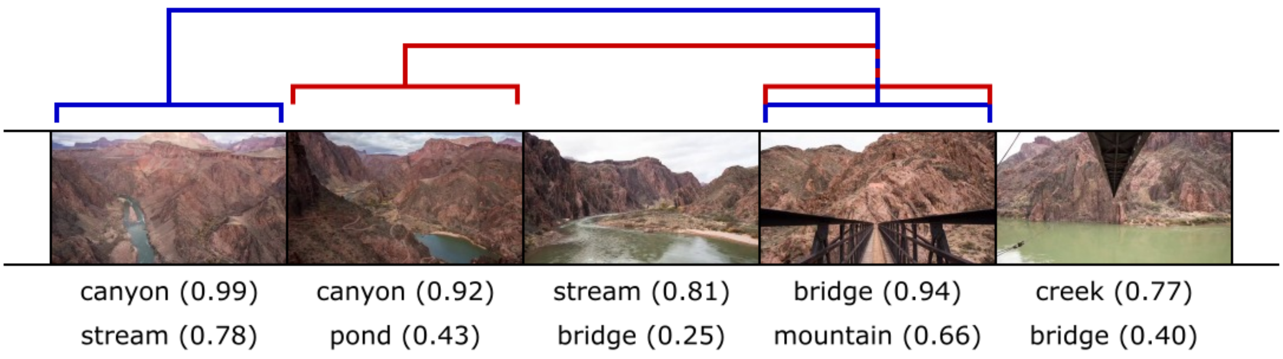

Fig. 2: Illustration of temporal sequences. Given a query with the semantic "canyon *occurs before* bridge", the blue temporal sequence — consisting of the first and fourth segment — is formed due to the third and fifth frame having an inferior score for "bridge". For the second temporal sequence — the red one — this applies as well. Since the first segment is already part of a temporal sequence, the second one is the starting point for the red sequence.

out containing any elements which match them all. In the first row of Figure 1, we show an example for this effect, as the row contains matches based solely on either (i) color not containing mountains or meadows, or (ii) on the tags, but not boosted by the sketch search. Another effect of this implicit filtering is that it is unable to explicitly remove elements from the final result list if they were retrieved by another module which was unaware of the filter criterion.

3. MULTI-STAGE QUERIES

To enable the use of explicit filtering between feature modules, we extend the vitrivr retrieval model by the notion of *query stages*. A query stage is simply a query which only processes elements which have been found to be relevant by another process, commonly a previous query stage. In order to keep the independence of the individual feature modules, the filtering for elements which are to be considered for a similarity query in the first place needs to be performed on an attribute which is common to all modules, independent of their operation. The only attribute matching this criterion is the *id* assigned to every element. We therefore slightly modify the process which needs to be performed by a feature module by the inclusion of a list of relevant ids as outlined in Listing 2. For most modules, this does not require any additional computational effort on the side of Cineast, since vitrivr's storage layer Cottontail DB (similar to its predecessor ADAM$_{pro}$ [6]) is capable of performing k-nearest-neighbour queries with additional Boolean filter constraints.

There are two areas of concern for performance: The first one being a loss of parallelism, since query execution for later terms have to wait for the full result list of their predecessor and the second one being the number of ids that have to be sent between storage layer and retrieval engine. We argue that both of these are negligible: The improved retrieval performance more than makes up for the loss of parallelism and early stages can usually be formulated as either Boolean or textual queries which have very low execution times, and as long as there exists some degree of selectivity in filter stages, the number of ids remains low enough for state-of-the-art datasets [1, 7] used at e.g., the Video Browser Showdown (VBS) [8] or the Lifelog Search Challenge (LSC) [9].

Users are able to move query terms up and down in the filtering order, with multiple terms per stage being allowed.

As shown in the second row of Figure 1, all results of multi-stage queries are based on both the tags and color.

```
SELECT id,
  score(
    feature_transform(query),
    feature_vectors
  ) AS score
FROM feature_table
WHERE id IN (...)
LIMIT 1000
ORDER BY score DESC
```

List. 2: Pseudo-SQL for per-feature vector space retrieval, retrieving the top 1000 relevant results limited to ids obtained by a previous process.

4. TEMPORAL SCORING

In recent installments of interactive multimedia retrieval evaluations, such as VBS or LSC, temporal queries gained attraction [10, 11, 12, 13] and might have even contributed their part to the success of the winner of VBS2020 [14].

As described in Section 2, the data model of vitrivr is constructed around *multimedia objects*, which consist of one (e.g., still images) or more (e.g., videos) *segments*. For query formulation, *query containers*, encapsulating one or more *query terms*, are used. Ultimately, a query is represented by an ordered list of such query containers. This explicit ordering can be mapped to a temporal order, where the order of elements in the query is supposed to reflect a temporal order to be considered in the query results. Our temporal scoring algorithm is based on the notion of *temporal sequences*, which consist of multiple segments corresponding to the query containers. Generally speaking, the algorithm is roughly inspired by the one the Vireo team proposed in [12].

From a high-level perspective, for a query with k query containers, we consider k segments to form a matching temporal sequence, if (a) they are from the same multimedia object, (b) occur within a certain time span, and (c) have the highest score for the corresponding query container, i.e., the first segment scores highest for the first query container and the last segment highest for the last query container. In order to detect such sequences, we use the following algorithm for each multimedia object:

1. The segments are sorted temporally, i.e., based on the timestamp in increasing order.
2. For each segment's scores, the initial temporal sequence is the segment itself. Then for each category within the segment's scores, potential temporal sequences are calculated. A potential temporal sequence is built incrementally using the current segment and the next segment, to which our definition of a temporal sequence can be applied. Ultimately, only those segments form the temporal sequence that result in the highest score, which is normalized to the number of query containers. This leads to multiple temporal sequences starting at a certain segment.
3. Finally, for each segment, only the temporal sequence with the highest score is kept and rendered as a result to the user.

In Figure 2, two examples of temporal sequences for the query "canyon *occurs before* bridge" are shown. This query is a possible vitrivr-specific representation of one of the textual known-item search tasks of VBS'19 [15] which was to find a segment showing "*A slow pan up from a canyon, static shots of a bridge and redrock mountain. A river is visible at the ground of the canyon.*". Based on the aforementioned algorithm, a first temporal sequence is built using the first and the fourth segment, as the fourth segment is the only one scoring for "bridge", resulting in the blue temporal sequence. A second sequence is built using the second segment and the fourth one, resulting in the red temporal sequence. Its noteworthy that the third segment is never considered, as it did neither score for "canyon" nor for "bridge". However, the fifth segment's score for "bridge" is inferior to the third segment's score, hence, a potential temporal sequence is dropped in the last step of the algorithm. In this example, it is also observable that a segment might occur in multiple sequences, if it fits a later query container.

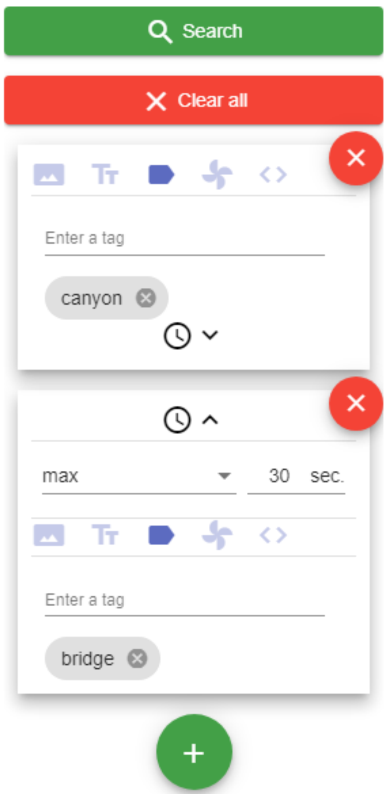

Fig. 3: The user interface for temporal queries. The depicted query asks for "canyon occurs before bridge within maximally 30 seconds".

In vitrivr's two step fusion model, introduced in Section 2, the second step is slightly adapted in the context of temporal scoring. In particular, the second step – the result fusion performed by the user interface – is adopted to the aforementioned algorithm for calculating temporal sequences. During query formulation, users add query containers per temporally distinguishable entity and, thus, explicitly indicate the temporal ordering. The maximal or minimal time span in between subsequent temporal entities is user configurable as well. In Figure 3, the user interface for temporal queries is shown with the example query of this section "canyon occurs before bridge". Additionally, the user specified, that there has to be at most 30 seconds in between the two matches. Upon result fusion, the temporal scoring view presents temporal sequences per multimedia object by visually indicating the segments, which belong to one sequence.

5. CONCLUSION AND OUTLOOK

In this paper, we presented the basic retrieval model of vitrivr, a full-stack open-source multimedia retrieval system. We introduce two new concepts in vitrivr, multi-stage and temporal queries. We hope that this paper can serve as a reference for other multimedia retrieval systems and inspire new discussions around the usefulness and implementation of both temporal and multi-stage queries. While temporal scoring has been used by multiple participants at VBS [10, 12], we think that there remain open questions with regard to query formulation, scoring algorithm, and results display. While many systems already use either Boolean filters or color-based filters in the retrieval process, we believe that multi-staged queries combined with sketch-based querying could prove to be very useful when retrieving from larger collections, as the selectivity of tag-based retrieval diminishes.

6. REFERENCES

[1] Luca Rossetto, Heiko Schuldt, George Awad, and Asad A Butt, "V3c–a research video collection," in *International Conference on Multimedia Modeling*. Springer, 2019, pp. 349–360.

[2] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li, "Yfcc100m: The new data in multimedia research," *Communications of the ACM*, vol. 59, no. 2, pp. 64–73, 2016.

[3] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt, "vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections," in *Proceedings of the 24th ACM international conference on Multimedia*, 2016, pp. 1183–1186.

[4] Luca Rossetto, Ivan Giangreco, and Heiko Schuldt, "Cineast: a multi-feature sketch-based video retrieval engine," in *2014 IEEE International Symposium on Multimedia*. IEEE, 2014, pp. 18–23.

[5] Ralph Gasser, Luca Rossetto, and Heiko Schuldt, "Towards an all-purpose content-based multimedia information retrieval system," *arXiv preprint arXiv:1902.03878*, 2019.

[6] Ivan Giangreco and Heiko Schuldt, "Adam$_{pro}$: Database support for big multimedia retrieval," *Datenbank-Spektrum*, vol. 16, no. 1, pp. 17–26, 2016.

[7] Cathal Gurrin, Klaus Schoeffmann, Hideo Joho, Bernd Munzer, Rami Albatal, Frank Hopfgartner, Liting Zhou, and Duc-Tien Dang-Nguyen, "A test collection for interactive lifelog retrieval," in *International Conference on Multimedia Modeling*. Springer, 2019, pp. 312–324.

[8] Klaus Schoeffmann, "Video browser showdown 2012-2019: A review," in *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*. IEEE, 2019, pp. 1–4.

[9] Cathal Gurrin, Klaus Schoeffmann, Hideo Joho, Andreas Leibetseder, Liting Zhou, Aaron Duane, Dang Nguyen, Duc Tien, Michael Riegler, Luca Piras, et al., "Comparing approaches to interactive lifelog search at the lifelog search challenge (lsc2018)," *ITE Transactions on Media Technology and Applications*, vol. 7, no. 2, pp. 46–59, 2019.

[10] Jakub Lokoč, Gregor Kovalčik, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech, "A framework for effective known-item search in video," in *Proceedings of the 27th ACM International Conference on Multimedia*, New York, NY, USA, 2019, MM '19, p. 1777–1785, Association for Computing Machinery.

[11] Jakub Lokoč, Gregor Kovalčík, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech, "Viret: A video retrieval tool for interactive known-item search," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019, pp. 177–181.

[12] Phuong Anh Nguyen, Jiaxin Wu, Chong-Wah Ngo, Danny Francis, and Benoit Huet, "Vireo@ video browser showdown 2020," in *International Conference on Multimedia Modeling*. Springer, 2020, pp. 772–777.

[13] Loris Sauter, Mahnaz Amiri Parian, Ralph Gasser, Silvan Heller, Luca Rossetto, and Heiko Schuldt, "Combining boolean and multimedia retrieval in vitrivr for large-scale video search," in *International Conference on Multimedia Modeling*. Springer, 2020, pp. 760–765.

[14] Miroslav Kratochvíl, Patrik Veselý, František Mejzlík, and Jakub Lokoč, "Som-hunter: Video browsing with relevance-to-som feedback loop," in *International Conference on Multimedia Modeling*. Springer, 2020, pp. 790–795.

[15] L. Rossetto, R. Gasser, J. Lokoc, W. Bailer, K. Schoeffmann, B. Muenzer, T. Soucek, P. A. Nguyen, P. Bolettieri, A. Leibetseder, and S. Vrochidis, "Interactive video retrieval in the age of deep learning - detailed evaluation of vbs 2019," *IEEE Transactions on Multimedia*, pp. 1–1, 2020.