

Evaluating Algorithms for Temporal Queries in Ad-Hoc Video Retrieval

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems
<https://dbis.dmi.unibas.ch/>

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: Silvan Heller, MSc.

Viktor Gsteiger
v.gsteiger@unibas.ch
2018-054-700

30.06.2021

Acknowledgments

Firstly, I would like to thank Prof. Dr. Heiko Schuldt for encouraging me to develop my computer science skills and allow me to write this thesis in the Databases and Information Systems Group. Furthermore, I would like to thank Silvan Heller for his valuable guidance that has lifted me and allowed me to see further and develop the skills necessary to conclude this thesis. Additionally, I also want to thank all the other group members and fellow students who have given valuable input and constructive criticism.

Lastly, I would like to thank the many people around me that helped me complete this thesis through their relentless and loving support.

Abstract

Expressing a temporal relationship between different search queries has become more important in recent years, especially when working with extensive video and audio data collections. Enabling queries with such a relationship is achieved by using temporal queries processed by temporal scoring algorithms. These algorithms aggregate the result sets of multiple search queries according to a temporal relationship and score the results regarding the similarity to the temporal query. In this thesis, seven such algorithms were developed and evaluated regarding response time and searched-item ranking as the primary metrics using a dataset of 109 queries specifically developed to test temporal query algorithms and specified in a newly developed format. The two best-performing algorithms were subsequently implemented in vitivr, a multimedia retrieval system, with changes to both the front-end and the back-end. The implementation was afterwards successfully used during a competitive evaluation of interactive multimedia retrieval systems. The competitive evaluation has shown that temporal querying in vitivr has noticeably improved with regards to response time and searched-item ranking due to the new algorithms and the new implementation within vitivr.

Table of Contents

Acknowledgments	ii
Abstract	iii
1 Introduction	1
1.1 Motivating Scenario	1
1.2 Multimedia Retrieval in vitivr	1
1.3 Temporal Multimedia Retrieval	2
1.4 Video Browser Showdown (VBS)	2
1.5 Contributions	2
1.6 Outline	3
2 Concepts	4
2.1 Retrieval Model of vitivr	4
2.2 Temporal Query	5
2.3 Scoring Algorithms	8
2.3.1 Simple Temporal Scoring (STA)	9
2.3.2 vitivr Temporal Scoring (VITRIVR)	10
2.3.3 A* Temporal Scoring (ASTAR)	12
2.3.4 Cluster Temporal Scoring (CLUSTER)	14
2.3.5 Inverse Decay Temporal Scoring (IDA)	15
2.3.6 Log Normal Temporal Scoring (LNA)	16
2.3.7 Normal Temporal Scoring (NA)	18
2.3.8 Sequential Temporal Scoring (SQA)	19
3 Evaluation	21
3.1 Setup	21
3.2 Evaluation Metrics	22
3.3 Evaluation Dataset	22
3.4 Temporal Efficiency	24
3.5 Qualitative Performance	29
3.6 Discussion	31
4 Implementation	34

4.1	Querying	34
4.2	Result View	36
4.3	Temporal Scoring	37
5	Related Work	39
5.1	VBS participants	39
5.1.1	Viret	39
5.1.2	Verge	40
5.1.3	Vireo	40
5.1.4	SOM-Hunter V2	41
5.1.5	W2VV++ BERT	41
5.1.6	Visione	42
5.2	Other Scoring/Rank Aggregation Algorithms	42
5.2.1	Prize-Collecting Steiner Tree Problem	42
5.2.2	Maximum Rank Aggregation Problems	42
5.2.3	Weighted Minimum Feedback Arc Set Problem in Tournaments	43
6	Conclusion	44
6.1	Conclusion	44
6.2	Further Work	44
	Bibliography	46
	Appendix A Task Specification	49
	Appendix B Evaluation Plots	56
	Declaration on Scientific Integrity	58

1

Introduction

Multimedia data, especially video and audio data, have a temporal context. Queries, where users express temporal context as part of their information needs, are called temporal queries in the context of this work. This thesis addresses this feature in vitivr by developing and evaluating new algorithms to retrieve results for a temporal query and subsequently implementing them.

1.1 Motivating Scenario

Let us introduce a small fictional scenario: Jane Doe is a documentary filmmaker with hundreds of hours of video stored in her multimedia database. The videos are from various projects and cover all sorts of topics. She has recently received a request for some stock footage from another filmmaker. She remembers having already captured the requested footage, a video where firstly a lion is sleeping, then a giraffe is eating leaves, and lastly, a couple of elephants take a bath in a lake. However, Jane does not remember when she had recorded this footage and does not remember any other information about the searched sequence.

She could use her retrieval engine to do a semantic search for the concepts of giraffes, lions, elephants individually and hope to find the correct result in a reasonable time frame by sheer luck and brute force searching.

Modern retrieval systems, however, can help Jane by introducing temporal queries. She can now search for the three segments and order them temporally to retrieve the wanted sequence.

1.2 Multimedia Retrieval in vitivr

The multimedia retrieval system utilized and extended during this thesis is vitivr [19] developed by the Databases and Information Systems group at the University of Basel. vitivr is an open-source, full-stack, content-based, multimedia retrieval system.

Multimedia retrieval in vitivr is based on abstraction layers used to index and retrieve the underlying multimedia data. Features can be computer-generated *semantic annotations*

but also *color* or *motion* information. Several similarity measures are employed to score a multimedia item in its similarity to a given query.

1.3 Temporal Multimedia Retrieval

As illustrated in the motivating scenario, temporal querying can be very productive in specific scenarios where someone searches for multimedia data with a temporal context. Temporal multimedia retrieval, which can also be described as a rank aggregation problem, involves aggregating results from multiple result sets retrieved by the individual similarity searches and scoring the aggregated results according to their similarity to the temporal query. Algorithms that do this aggregation and subsequently score the results will be called temporal scoring algorithms. This thesis will develop seven and evaluate eight temporal scoring algorithms, with the current implementation in *vitivr* as the baseline, to enable fast and efficient temporal multimedia retrieval.

1.4 Video Browser Showdown (VBS)

The Video Browser Showdown (VBS) [22] is an annual competition of multimedia retrieval systems to evaluate the abilities of the participating systems competitively.

The competitors all use the same dataset; in recent instalments, the utilized dataset was the *Vimeo Creative Commons Collection* (V3C) [20]. The VBS involves two types of tasks to evaluate the systems:

- **Known-Item Search (KIS):** A single video clip of varying length must be retrieved from the collection. This task has two variations, one where the known item is displayed on a central screen and one where the moderators textually describe the known item.
- **Ad-hoc Video Search (AVS):** A general description of a collection of shots is presented, and the goal is to submit as many shots as possible that fit the description. Whether the shots fit the description is judged by a jury.

The contribution of this thesis with regards to the VBS competition is to develop a more efficient temporal querying to give *vitivr* a competitive advantage over other retrieval systems with regards to the KIS tasks.¹

1.5 Contributions

The contributions of this thesis are the following:

- Develop several temporal scoring algorithms
- Define an evaluation task specification and establish a set of evaluation tasks and corresponding temporal queries that can be used to evaluate temporal scoring algorithms

¹ AVS tasks have not been specifying temporal sequences in the past. However, one can easily imagine such tasks.

- Evaluate the quality of these temporal scoring algorithms concerning the given temporal queries.
- Transfer the current temporal scoring aggregation and scoring of vitrivr from the front-end to the back-end and implement the temporal scoring algorithms determined to be the best.
- Present the results on the front-end in an appealing fashion

1.6 Outline

Firstly, this thesis will discuss relevant concepts and the conceptual contribution of the developed temporal scoring algorithms in Chapter 2. In Chapter 3, the evaluation of the algorithms will be discussed, while in Chapter 4, the implementation into the existing systems will be introduced, and related work will be discussed in Chapter 5. The conclusion and an outlook on possible future contributions will be presented in Chapter 6.

2

Concepts

In the following sections, firstly, the retrieval model of vitrivr will be introduced. Following this, this thesis’s conceptual contributions concerning temporal queries and scoring algorithms will be discussed.

2.1 Retrieval Model of vitrivr

This thesis will regularly refer to the vitrivr retrieval model. This section is based on what Heller et al. mentioned [8]. The three core components of the vitrivr system are a multimedia retrieval database Cottontail DB [7], the retrieval engine Cineast [17], and the browser-based user interface vitrivr-ng [6]. The entire stack is open source and can be found on Github². The vitrivr retrieval stack provides content-based retrieval functionality with multiple query modalities and similarity notions. The frequently used query modalities in this thesis are the textual search modalities with *Automatic Speech Recognition* (ASR), *Optical Character Recognition* (OCR), *Semantic Tags* and *Visual Text Co-Embedding*. In order to provide the multiple modalities and similarity notions, the query processing component — Cineast — provides a runtime for an arbitrary number of independent *feature models* which can selectively be used depending on the properties of the queries. The independent feature modules can perform any database retrieval operation, with the only constraint being that they produce a scored list of the most similar items regarding the query. Most modules perform vector space retrieval. In vector space retrieval, the documents and queries are represented as vectors. The similarity of a query and a document is then expressed as the similarity of the two vectors. Because vitrivr employs multiple independent retrieval modules, the types of similarity queries over multimedia data can be extended and adapted to the current application [6]. A *similarity query* in Cineast is defined as a query that should retrieve a list of scored items from a multimedia data collection with a similarity score.

The feature modules provided by vitrivr receive a query representation as the API provided by Cineast is defined. The API defines a data model for queries to the Cineast retrieval engine consisting of a list of several *query containers*. Each query container, in the context

² <https://github.com/vitrivr>

of this thesis also understood to be synonymous with similarity query, can consist of several stages with their respective *terms*, with each describing the content of a different modality, such as the previously mentioned semantic, visual, or auditory and many more. The staged queries are applied in subsequent order, where each result set represents a filter for the following similarity query. The modules independently perform similarity searches based on the components of the query that are important for them. The results produced by the feature modules are then aggregated using a two-step score-based weighted fusion scheme. The first step is performed by Cineast, where groups of feature modules with similar notions of similarity are aggregated. The second step of the scheme is performed by vitrivr-ng based on interactive configurable weights for the different components. As expected, the fusion process encourages results scored by multiple modules to be superior to those scored by just one module. Features that are not scored by any module receive a score of 0; the modules return a list of all relevant elements scored with the highest possible score of 1.

The contribution of this thesis will extend the existing score-based fusion scheme with a temporal aggregation and scoring scheme. The existing fusion process will not be adapted to enable backwards compatibility fully.

2.2 Temporal Query

As discussed in the previous section, the current query data model enables the formulation of temporally chained query containers representing a staged similarity query. The chained staged similarity queries are then supposed to be scored to aggregate the temporal sequences desired by the user. The scoring process is supposed to boost the scores of objects with more and higher results in multiple such containers. The mentioned problem can also be described as a rank aggregation problem. The assigned scores that rank results within multiple result sets must be aggregated according to a temporal sequence to retrieve the elements that best describe the temporal sequence.

A temporal query will be defined by making the following assumptions:

- A temporal query container denoted t_i with $i \in \mathbb{N}_0$ represents a similarity query as defined by the retrieval model of Cineast 2.1
- A temporal query consists of one or more temporal query containers $\{t_1, \dots, t_n\}$ for $n \geq 1$
- A temporal query has an absolute order from the first to the last segment of the sequence with $t_i < t_j$ for all $i < j$, $i, j \in \mathbb{N}_0$,
- Only multimedia objects that can be splittet up into segments of data that have a temporal order are considered for the results of a temporal query
- Temporal scoring creates a list of tuples $\langle R_{sequence}, s_{sequence} \rangle$ with $R_{sequence} = \{r_1, \dots, r_i\}$ for $n \geq i \geq 1$ a sequence of segments from the multimedia data collection and $s_{sequence} \in \mathbb{R}_0$ a similarity score,

- Each result segment $r_i \in R_{sequence}$ has to correspond to a query container t_i ,
- Two results $r_i, r_j \in R_{sequence}$ where $i \neq j$ and $i < j$ have to be in the same order as the corresponding query containers t_i, t_j where $i \neq j$ and $i < j$,
- The result of a temporal query is then a list of tuples $\langle R, s \rangle$ with R being a set of all segments from the same object with unique values from the list of tuples created by temporal scoring and s being the maximum score from all sequences corresponding to an object from the list of tuples created by temporal scoring.

For a further illustration, the following graphs will illustrate the beforementioned concepts with the illustrative example from Section 1.1:

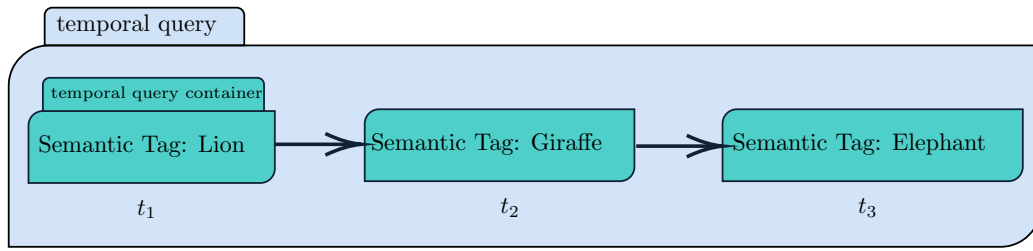


Figure 2.1: A temporal query.

A temporal query as shown in Fig. 2.1 should result in a temporal sequence. A temporal sequence is based on the aggregation and scoring of the different result sets from the retrieval containers. Whether a *sequence* of results from multiple containers is defined as a better temporal sequence than another depends on how well the aggregated results reflect the temporal query. The more segments in the correct absolute order of the query and the more segments present in multiple container result sets, the higher the score of a retrieved temporal sequence should be. Additionally, it may be that the system allows the user to define time distances between two segments. The closer the two segments time difference is to the one provided to the user, the higher the score of the entire temporal sequence should be. The scoring should be based on these measures. The exact implementation of this aggregation score depends on the unique algorithm.

The following example of a result from a temporal query will further illustrate the points discussed. The scores and created sequences do not represent the results of a specific algorithm and are intended for illustration only. Furthermore, the identification used in the following examples consists of `v_objectId.segmentId`.

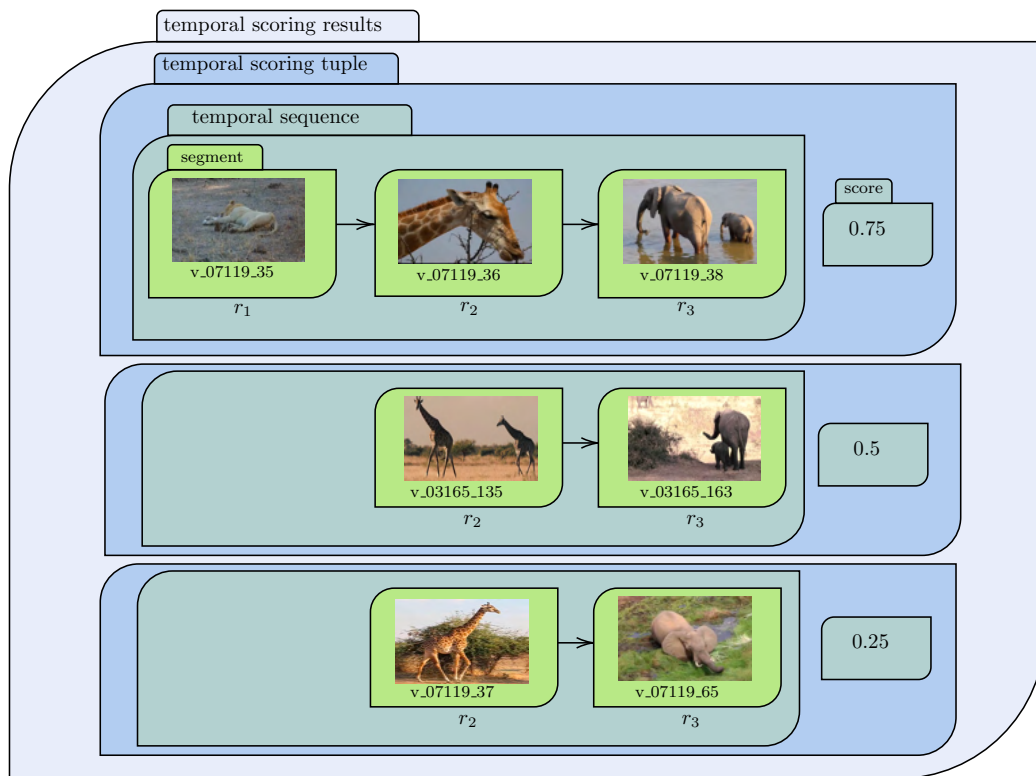


Figure 2.2: A temporal scoring result with the sequences generated over all objects by some scoring algorithm.

The temporal scoring result illustrated in Fig. 2.2 is a potential result list from the query formulated in Fig. 2.1. Three different result tuples can be seen in the list, two from the same object with different sequences and one from another object. Fig. 2.3 will display how the result of a temporal query would look like given the scoring results just mentioned.

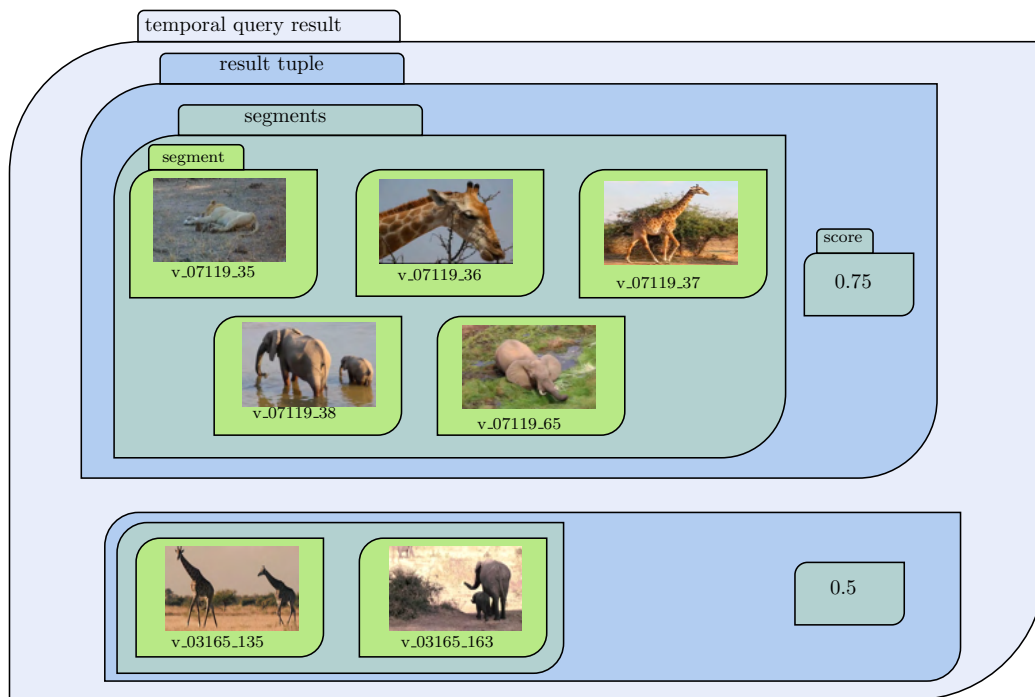


Figure 2.3: A temporal query result where the segments contributing to the sequences have been mapped to their respective object and the score of the best sequence is assigned to its object.

The segments in the result tuples should be ordered by occurrence within an object, as can be seen in Fig. 2.3. Here also, the max-pooling applied to each object can be seen. Max-pooling means to take the highest value of a set to describe the set. In the subsequent section, the developed temporal scoring algorithms will be applied to the example query results from Fig. 2.2. In Chapter 4 implementing the different steps elaborated in this section will be discussed.

2.3 Scoring Algorithms

In the following subsections, the temporal scoring algorithms developed during this thesis will be introduced. Additional ideas for temporal scoring algorithms will be discussed in Chapter 5.

All algorithms expect one or more result containers as input, with each container consisting of key-value pairs representing the segment ids of the multimedia data and the retrieval engine score. The total order of the containers represents the temporal ordering of the aggregated sequences.

The algorithms return temporal sequences with the object id of the corresponding object, the start and end of the retrieved temporal sequence and the score given to the temporal sequence.

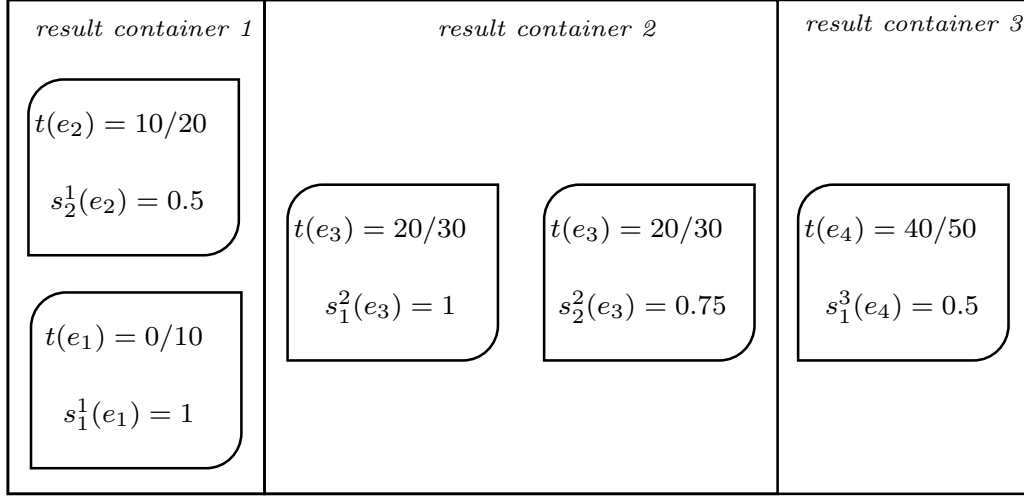


Figure 2.4: Example segment results.

Fig. 2.4 shows an illustrative example that will be used with all of the following algorithms. Five result segments will be focused on from the same object as all the algorithms apply their logic the same way within one object. In the example $t(e_i) = \text{startTime}/\text{endTime}$ denotes the start and end of an element e_i and $s_j^c(e_i) = \text{score}$ denotes the score of element e_i for query container c where it is the j^{th} result for that query container. If an element has no score for a query container, then the retrieval engine did not retrieve this element for this query container. The provided time distances for the query are $T = [10, 10]$ where there should be a distance of 10 seconds between the results of the first two containers and again 10 seconds between the results of the second and third container.

2.3.1 Simple Temporal Scoring (STA)

The Simple Temporal Scoring Algorithm scores the retrieved results according to a sequence of steps in which results, that return the sought after segments in the correct order and inside a given time limit, are connected to a temporal sequence. The steps performed during the scoring are the following:

1. For each temporal query result container, the retrieved results get accumulated into similarity result storages if more than one result has been retrieved for a segment.
2. For each object, all the similarity result storages are retrieved ordered based on the segment ids present within the storages. The storages are sorted in the temporal order of the start of the corresponding segment.
3. The algorithm walks through every similarity result storage and constructs temporal sequences based on the current segment.
4. To construct the temporal sequences, the highest-scoring similarity result in the next container within the defined time distance is retrieved and added to the sequence.
5. The sequence for every similarity result storage with the highest score is saved.

6. The temporal sequences are extracted from the store of the temporal sequences with the highest score. The score is normalised over the number of temporal result containers.

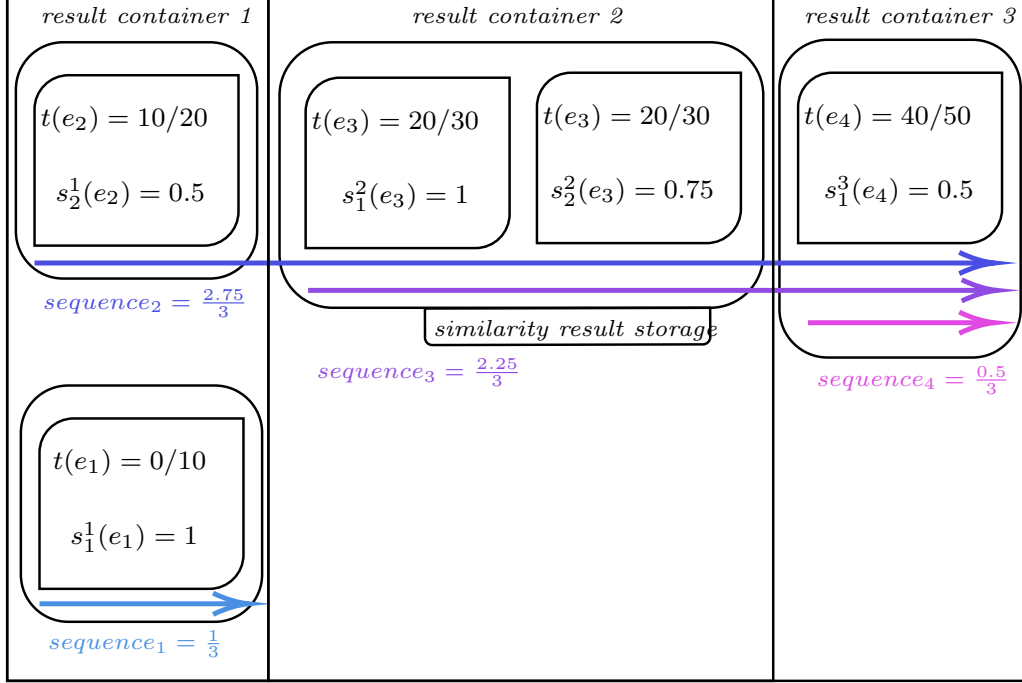


Figure 2.5: Example temporal scoring by simple temporal scoring.

When STA is applied on the example results introduced at the beginning of this section, the algorithm retrieves four temporal sequences as seen in Fig. 2.5 with *sequence₂* being the one with the highest score. Blue arrows note the sequences kept for each segment. How the similarity result storage has been created can be seen, following it can be seen how the algorithm constructs the temporal sequences going out from every similarity result storage. *sequence₁*, for example, is so short because the distance to the next segment from another container is too far away to be connected.

2.3.2 vitivr Temporal Scoring (VITRIVR)

A temporal scoring algorithm has already been implemented in the instance of vitivr present before starting this thesis. The description of the algorithm is based on what was mentioned by Heller et al. [8]. The steps performed during the scoring are the following:

1. The multimedia objects segments are sorted temporally, usually based on timestamps in increasing order.
2. Initially, each segment receives its score, and the initial temporal sequence is the segment itself. Then for each category within the segment's scores, potential temporal sequences are constructed. The construction of potential temporal sequences happens

incrementally using the current segment, and the next segment that fulfils the definition of a temporal sequence is combined with a temporal sequence. Only those segments form the temporal sequence that results in the highest score, which is normalised to the number of query containers. This process leads to multiple temporal sequences starting at a certain segment.

3. For each segment, only the temporal sequence with the highest score is kept, normalised over the number of temporal containers and rendered as a result.

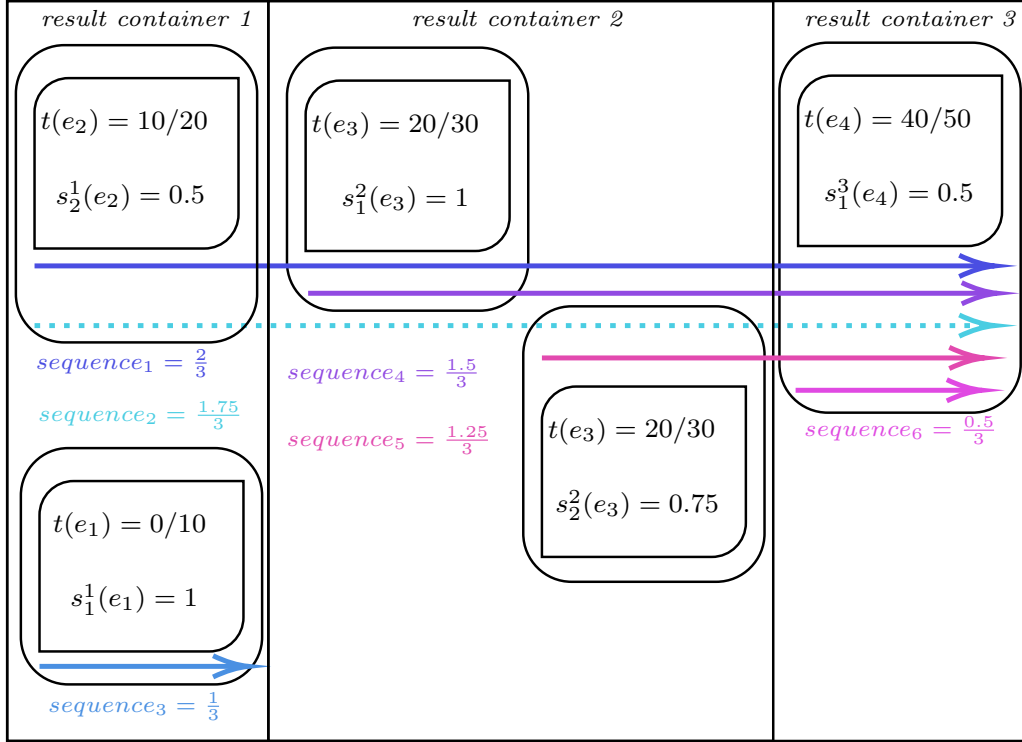


Figure 2.6: Example temporal scoring by vitivr.

When vitivr temporal scoring is applied on the example results introduced at the beginning of this section, five temporal sequences are retrieved as seen in Fig. 2.6 with $sequence_2$ being the one with the highest score. Blue arrows note the sequences kept for each segment. A segment was scored on its own and not combined as the algorithm created temporal sequences for all categories independently. In this example, the assumption that $s_1^2(e_3)$ and $s_2^2(e_3)$ come from different categories that are both present with the same score in all other results was made. This distinction leads to two result sequences going out from segment e_2 , but only the higher one is kept. Additionally, $sequence_1$, for example, is so short due to the same reason as previously discussed with STA.

2.3.3 A* Temporal Scoring (ASTAR)

The idea for an A* temporal scoring algorithm was developed while taking the lecture Introduction to Artificial Intelligence³, where several heuristic planning algorithms were introduced.

The application utilised for temporal scoring of an A* algorithm represents the temporal layout of a multimedia query as a tree structure. The tree nodes are all the segment scores added up. The initial node is in each tree one segment to start with. Each layer of the tree is one temporal query container. The ordering of the nodes from left to right is in sequential order of the queries.

The heuristic applied indicates how well the following video fits into the temporal sequence. So the heuristic penalises segments that are to the left of the current node (before), assigns a linearly increasing score between the current segment and the correct time segment, and segments outside the given boundary. The action cost is the time distance between the current node and the following node where a perfect result has distance 1. Every node with an absolute time distance equal to or less to the perfect result receives a cost of the perfect time distance subtracted by the actual time distance. Every node with an absolute time distance larger than the perfect result receives the cost of 150 added with the absolute time difference between perfect and actual.

```

1 open := new MinHeap ordered by (f, h)
2 if h(init()) < infinity:
3     open.insert(make_root_node())
4 distances := new HashTable
5 while not open.is_empty():
6     n := open.pop_min()
7     if distances.lookup(n.state) = none or g(n) < distances[n.
        state]:
8         distances[n.state] := g(n)
9         if is_goal(n.state):
10            return extract_path(n)
11        for each (a, s') element of succ(n.state):
12            if h(s') < infinity:
13                n' := make_node(n, a, s')
14                open.insert(n')
15    return unsolvable

```

Listing 2.1: Generic A* psuedo-code.

The A* algorithm can be best described by pseudo-code as seen in Listing 2.1. The goals reached in our example are the optimal temporal sequences that then get normalised over the number of temporal containers. In our example, the function is $f = g + h$ where g

³ <https://dmi.unibas.ch/de/studium/computer-science-informatik/lehrangebot-fs21/lecture-foundations-of-artificial-intelligence/>

is the distance of a node to the root along the current path. Additionally, a penalty for segments outside the defined temporal sequence is added. The heuristic h employed for our implementation of the A* temporal scoring algorithm is the following:

$$h(s) = \begin{cases} (1 - \frac{s.current.end - s.previous.end}{s.timeDistance}) \cdot (s.nQueriesLeft), & \text{if } s.previous.end \leq s.current.start \leq s.previous.end + s.timeDistance \\ 150, & \text{otherwise} \end{cases} \quad (2.1)$$

And the cost of an action is calculated as follows:

$$cost(a, timeDistance) = \begin{cases} timeDistance - a.moveDistance + 1, & \text{if } a.moveDistance \leq timeDistance \\ 150 + |timeDistance - a.moveDistance + 1|, & \text{otherwise} \end{cases} \quad (2.2)$$

A* with reopening that was employed during the development of this algorithm is optimal when using an admissible heuristic. To show that the heuristic is admissible, it is necessary to prove that the employed heuristic is consistent and goal-aware. Goal-awareness is proven easily since the only time our heuristic has the value 0 is at a goal state when the number of queries left is 0.

Consistency can be shown by looking at the triangle inequality $h(s) \leq cost(a) + h(s')$ for all transitions $s \xrightarrow{a} s'$:

- From the initial state the heuristic value $h(s)$ will always be 1 if there are any following nodes and 0 if the initial state is the goal state. Any action cost will be at least 1, so $1 \leq 1 + h(s')$ or $0 \leq 1 + h(s')$ will always be true.
- For a state where the algorithm previously stepped outside the desired time distance, the algorithm will get a heuristic value of $h(s) = 150$ but in the following state s' the algorithm might step into the desired next time distance and get both a heuristic value $h(s')$ and $cost(a)$ lower than 150. However, the algorithm accepts this because it encourages correct sequences aside from the correct time distance.
- If the algorithm previously stepped inside the desired time distance, the algorithm will get a value between 0 and 1 times the number of temporal containers left. The cost of any action will be 1 and thus $x \leq 1 + h(s')$ for $x \in [0, 1]$ will be true.

Therefore our heuristic is not consistent and thus also not admissible and safe. Consequently, A* may find only suboptimal solutions. However, during development, the current implementation was considered to be sufficient due to its logical setup and the encouraging design that boosts results with a perfect time distance even if a previous segment was outside the desired time distance. In a future improvement, it could be interesting to change the cost function to reflect the number of containers left. The utilised heuristic function also has the consequence that every state has to save its time value and previous time value and the time distance at the current level. This design is a slight deviation from the original state space problem as states usually don't store information regarding predecessors. However, as this information is only utilised to calculate the heuristic value, this deviation should not influence the optimality of the heuristic.

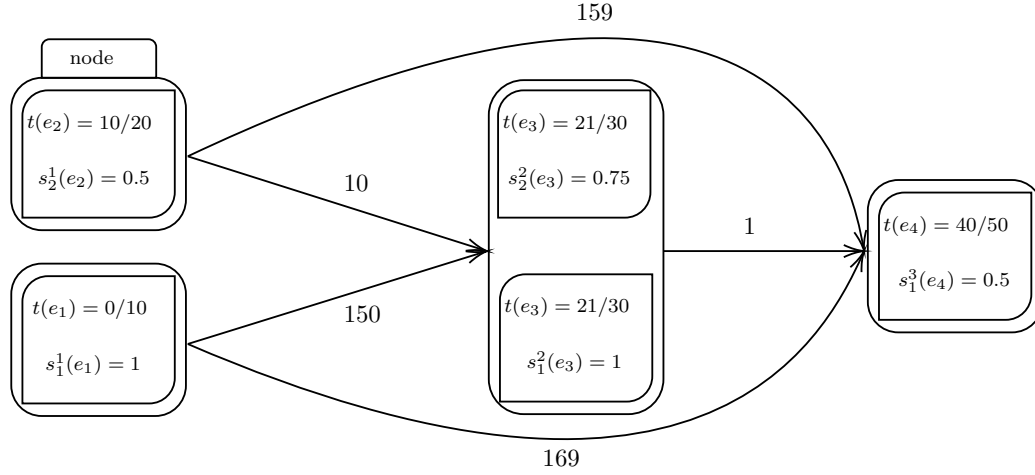


Figure 2.7: State space created by A* search.

To illustrate A* the state space from the previously discussed example as seen in Fig. 2.7 has been depicted. The heuristic values of the different states could then easily be calculated.

2.3.4 Cluster Temporal Scoring (CLUSTER)

The cluster temporal scoring algorithm creates clusters of one to several segments. The algorithm creates clusters if two segments have been scored within the same container and if they are temporally close according to a distance measure. The borders of the clusters are adapted to the maximum and minimum value of their contained segments. This creation of clusters happens within each temporal container.

1. Cluster objects are created by accumulating all results within a boundary of 2 seconds and if both results have received a scored result within one result container.
2. The cluster objects get sorted temporally within their query.
3. Then, for each cluster, potential temporal sequences are constructed. This construction happens incrementally using the current cluster, and the next cluster that fulfils the definition of a temporal sequence is combined with a temporal sequence.
4. For each cluster, only the temporal sequence with the highest score is kept, normalised over the number of temporal containers and rendered as a result.

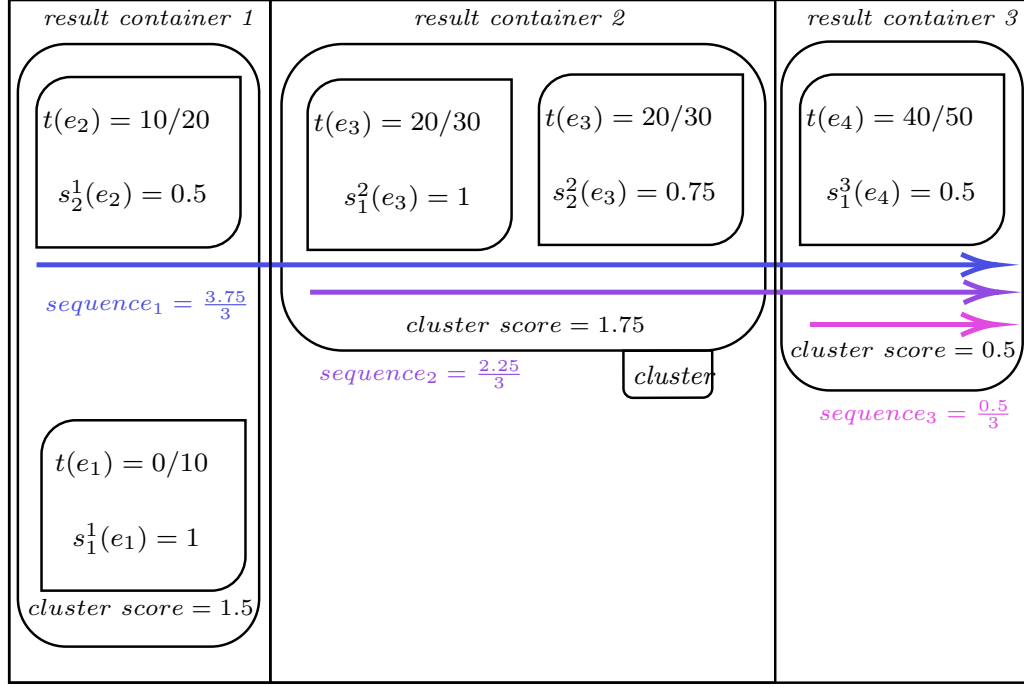


Figure 2.8: Example temporal scoring by CLUSTER.

When CLUSTER is applied on the example results introduced at the beginning of this section, three temporal sequences are retrieved as seen in Fig. 2.8 with $sequence_1$ being the one with the highest score. Blue arrows note the sequences kept for each segment. The first two segments have been combined into a cluster because they are temporally close to each other. This creates a new sequence not seen before with a higher score. However, potential temporal sequences can be lost due to clustering compared with the other algorithms.

2.3.5 Inverse Decay Temporal Scoring (IDA)

The inverse decay temporal scoring algorithm attempts to implement an algorithm similar to the one currently employed by vitivr. The difference between the two is that inverse decay significantly boosts results in the correct time sequence while largely penalising results that do not adhere to the time differences while not completely discounting results that are not in the correct time location but closer. Results that are farther away than the perfect result will be penalised strongly. At the same time, the inverse decay temporal scoring algorithm employs iterative methodologies instead of recursive ones. The steps the algorithm performs are the following:

1. Each segment receives its score calculated by adding up the individual scores from the similarity result. The initial temporal sequence is the segment itself.
2. Then, for each query within the segment's scores, potential temporal sequences are constructed. This construction happens incrementally. The current segment and the next segment that fulfils the definition of a temporal sequence are combined into a temporal sequence. While combining the results, the weighing of the scores is done

with a decay function. Segments between the current segment and the time to the next segment get their score-adjusted with $adj(t) = e^{l(t-m)}$ with $l > 0$ being the penalty of being not at the perfect spot, $m \leq 0$ being the time defined to the next segment, and $t > 0$ being the time difference between the next element start and the current element end. The scores of the segments after the time defined for the next segment will be adjusted by $adj(t) = e^{-l(t-m)}$ with $l > 0$ being the penalty of being not at the perfect spot, $m \leq 0$ being the time defined to the next segment, and $t > 0$ being the time difference between the next element start and the current element end. In our application $l = 0.1$ is employed in both instances. Items that are within the perfect time distance receive no penalty on their score.

3. Only those segments, that results in the highest score, which is normalised to the number of query containers, form the temporal sequence. This process leads to multiple temporal sequences starting at a particular segment.
4. For each segment, only the temporal sequence with the highest score is kept, normalised over the number of temporal containers and rendered as a result.

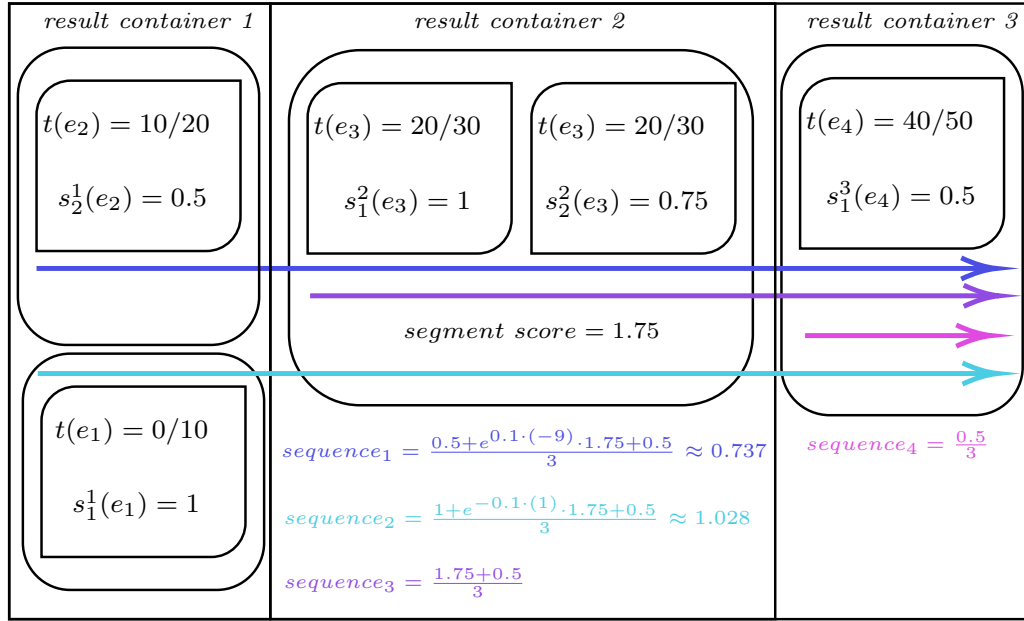


Figure 2.9: Example temporal scoring by IDA.

When IDA is applied on the example results introduced in the beginning of this section, four temporal sequences are retrieved as seen in Fig. 2.9 with sequence_2 being the one with the highest score. A more fuzzy definition of the temporal distances is creating sequences that have not been created compared with STA from Section 2.3.1.

2.3.6 Log Normal Temporal Scoring (LNA)

The log-normal temporal scoring algorithm is very similar to the inverse decay while employing a different weighting function to penalise the non-perfect results. The actual im-

plementation also varies between the two algorithms in the employment of data structures. The steps the algorithm performs are the following:

1. Each segment receives its score calculated by adding up the individual scores from the similarity result. The initial temporal sequence is the segment itself.
2. Then, for each query within the segment's scores, potential temporal sequences are constructed. This construction happens incrementally using the current segment and the next segment that fulfils the definition of a temporal sequence will be combined into a temporal sequence. While combining the results, the weighing of the scores is done with a log-normal function. The following equation penalises the score:

$$f(x) = \frac{1}{(-x + m) \cdot s \cdot \sqrt{2 \cdot \pi}} \cdot \exp\left(\frac{-\ln(-x + m)^2}{2 \cdot s}\right) \quad (2.3)$$

Where m is the input time difference plus 0.6, s is a constant, with $s = 0.5$ in our application, and x is the absolute actual time difference between the segments. As can be seen according to the function, results that are farther away than +0.6 from the perfect score will be set to zero as the function is undefined for $x \geq m + 0.6$. These variable numbers have been chosen according to intuition based on trialling out different versions.

3. Only those segments, that result in the highest score, which is normalised to the number of query containers, form the temporal sequence. This process leads to multiple temporal sequences starting at a particular segment.
4. For each segment, only the temporal sequence with the highest score is kept, normalised over the number of temporal containers and rendered as a result.

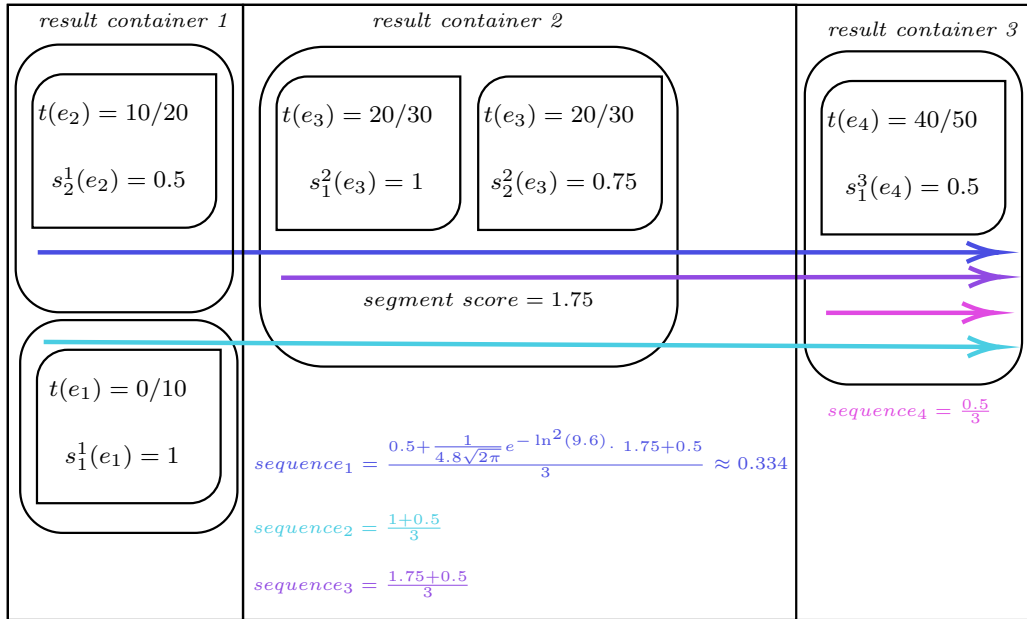


Figure 2.10: Example temporal scoring by LNA.

When LNA is applied on the example results introduced at the beginning of this section, four temporal sequences are retrieved as seen in Fig. 2.10 with *sequence₃* being the one with the highest score. As expected, the results are similar to previously discussed IDA in Section 2.3.5. However, border cases could be imagined where LNA would cut off temporal sequences for being too far away while IDA keeps them.

2.3.7 Normal Temporal Scoring (NA)

The log-normal temporal scoring algorithm is very similar to the inverse decay while employing a different weighing function to penalise the non-perfect results. The actual implementation also varies between the two algorithms in the employment of data structures. The steps the algorithm performs are the following:

1. Each segment receives its score calculated by adding up the individual scores from the similarity result. The initial temporal sequence is the segment itself.
2. Then, for each query within the segment's scores, potential temporal sequences are constructed. This construction happens incrementally using the current segment, and the next segment that fulfils the definition of a temporal sequence, combining them into a temporal sequence. While combining the results, the weighing of the scores is done with a normal function. The following equation penalises the score:

$$f(x) = 5 \cdot \frac{1}{2 \cdot \sqrt{2} \cdot \pi} \cdot \exp\left(\frac{-1}{2} \cdot \frac{x - m^2}{2}\right) \quad (2.4)$$

Where m is the time, input time difference and x is the actual absolute time difference.

3. Only those segments, that result in the highest score, which is normalised to the number of query containers, form the temporal sequence. This process leads to multiple temporal sequences starting at a particular segment.
4. For each segment, only the temporal sequence with the highest score is kept, normalised over the number of temporal containers and rendered as a result.

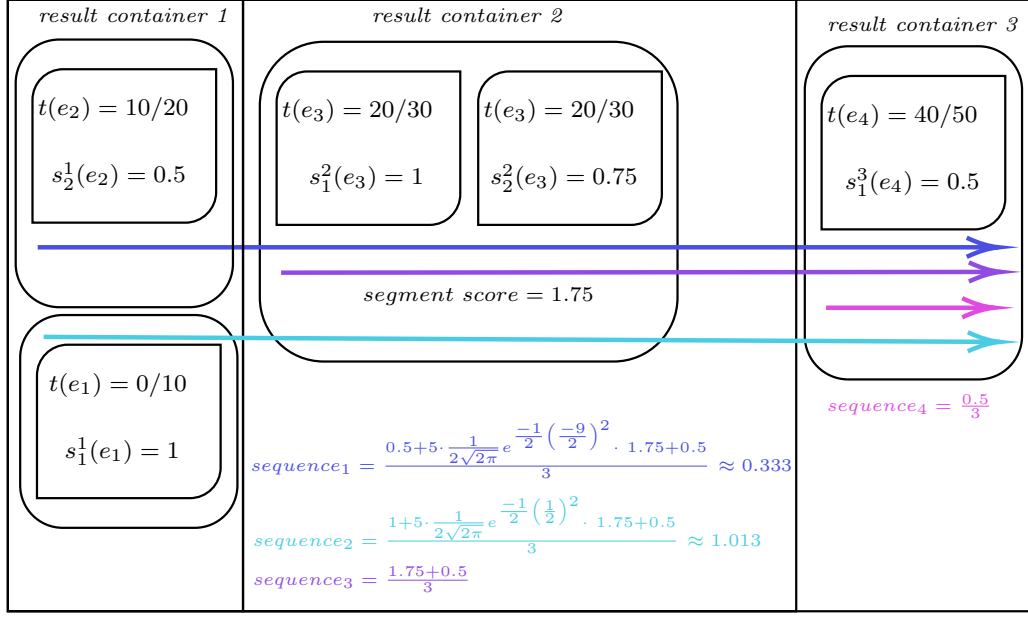


Figure 2.11: Example temporal scoring by NA.

When NA is applied on the example results introduced at the beginning of this section, four temporal sequences are retrieved as seen in Fig. 2.11 with *sequence₂* being the one with the highest score. As expected, the results are similar to previously discussed LNA in Section 2.3.6. However, there are cases where NA will penalise slightly differently than LNA or IDA where NA will boost a different sequence compared to the others.

2.3.8 Sequential Temporal Scoring (SQA)

Sequential temporal scoring algorithm differs from the previously discussed algorithms. The significant difference to all previously discussed algorithms is that the sequential temporal scoring algorithm does not care about any user input regarding the time distance between two temporally subsequent segments. It instead checks for the existence of a sequence in general. It, therefore, boosts segments that have results in all temporal containers and of which the results are in the correct order.

1. The sequential temporal algorithm goes through all extracted segments and constructs temporal sequences in the right order.
2. It calculates the best score using a priority queue to build and evaluate possible temporal paths from the first segment to the last.
3. For each segment, only the best temporal path is kept, and its score is normalised over the number of temporal queries.
4. For each segment, the best path is set as the temporal sequence to be ordered by descending score.

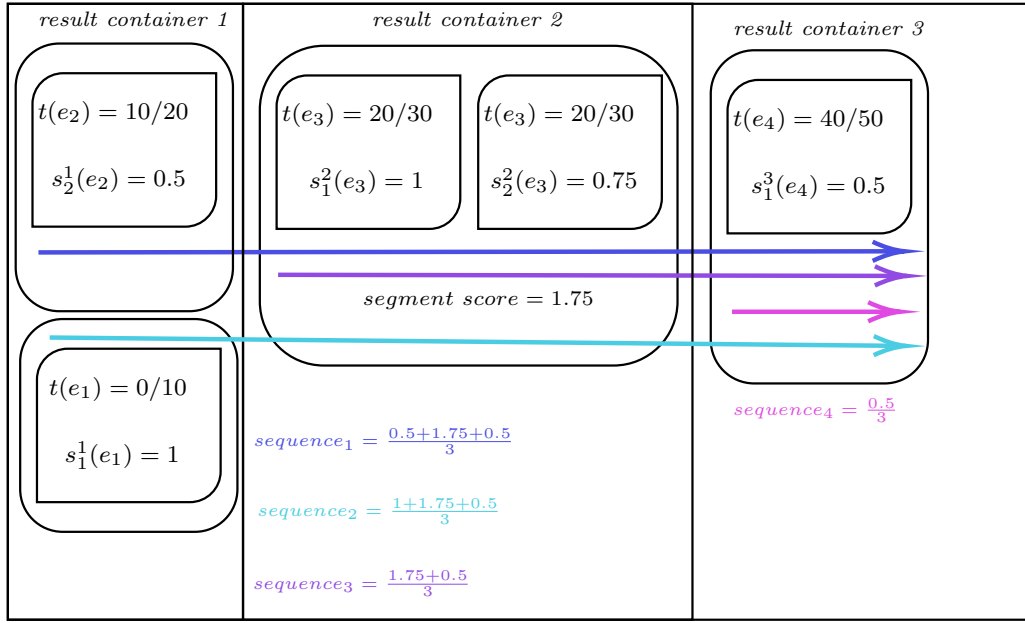


Figure 2.12: Example temporal scoring by SQA.

When SQA is applied on the example results introduced at the beginning of this section, four temporal sequences are retrieved as seen in Fig. 2.12 with $sequence_2$ being the one with the highest score. As expected, the results are similar to previously discussed algorithms. However, no score adjustments were applied as only the sequence counts and the time distances are ignored.

3

Evaluation

In this chapter, the algorithms presented in Section 2.3 will be evaluated. All the specifications⁴ for the evaluation are provided, and the source code is public so that others may replicate the results or test out their algorithms. First, the evaluation setup will be introduced. Following this, the evaluation metrics employed in general will be proposed. Subsequently, the evaluation dataset will be examined, followed by a discussion of the results.

3.1 Setup

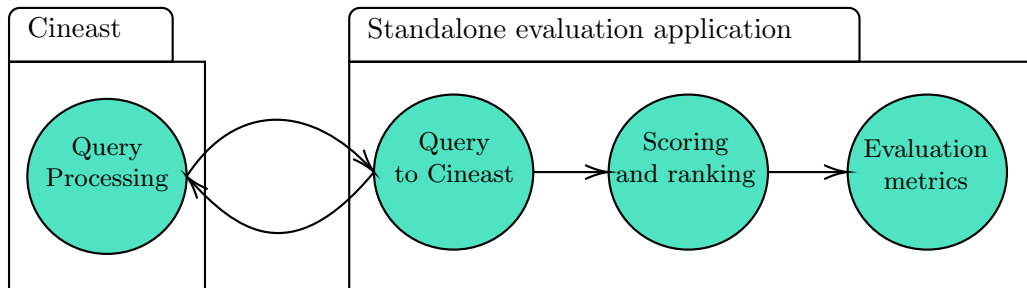


Figure 3.1: The standalone application setup.

A test setup of the vitivr stack evaluated the different temporal scoring algorithms. The evaluation process can be seen in Fig. 3.1. For the evaluation, a Cottontail DB instance provides the application’s database, Cineast provides the querying and the application interfaces, and the standalone evaluation application⁵ developed for this thesis provides the environment to test the algorithms. Vitivr-ng was therefore replaced with the evaluation application. The evaluation application implements the interfaces as described in Section 2.3 and ensures that all algorithms have the same framework for their evaluation.

The evaluation application queries Cineast with the WebSocket interface and then scores the

⁴ <https://github.com/vGsteiger/BachelorsThesisContribution/blob/main/Resources/specifications.json>

⁵ <https://github.com/vGsteiger/BachelorsThesisContribution/tree/main/tempAlgEval>

results returned by Cineast with the different temporal scoring algorithms. Then the evaluation application creates evaluation reports in the CSV format based on several evaluation metrics discussed in the following chapters. Kotlin was used to implement all algorithms. Similar data structures, for example, HashMaps and ArrayLists, were employed while developing the algorithms to ensure no structural disadvantages.

The algorithms were evaluated on a 2017 MacBook Air running with macOS Big Sur 11.2.3 and a 1.8 GHz Dual-Core Intel Core i5 processor and 8 GB 16000 MHz DDR3 Memory. The Cottontail DB instance was run on an external hard drive, a Seagate BarraCuda Fast SSD with 500GB storage.

3.2 Evaluation Metrics

A temporal scoring algorithm in an interactive multimedia retrieval setting has to provide multiple desired properties to help the user. The scoring algorithm has to be fast enough, and the algorithm has to provide good results. In the specific setting of this thesis, it should also improve performance relative to the current implementation. Therefore, the following two metrics will be evaluated:

Speed with regards to response time is measured by the Kotlin native `measureNanoTime`⁶ function, which is used around the scoring interface of the evaluation application to execute the interface function to score and returns the elapsed time in nanoseconds. Interesting is not only a fast execution in some queries but rather a constantly similar performance within a reasonable time.

Quality of result sets is in the scope of this thesis defined as the relative and absolute position of the known item within the result set returned by the scoring algorithm. To be more precise, the known item position is calculated by counting the position of the first temporal sequence with the same object ID as the correct result that is within a 15 seconds boundary of the correct item times. The boundary is defined so that the retrieved result has to start within a 15-second radius of the beginning of the correct result start time and ends within a 15-second radius of the correct result end time. Furthermore, the mean and standard deviation and percentage of results within relevant ranges of result sets are additional qualitative metrics evaluated in the following Sections.

Precision-recall plots were also created during the evaluation process, as this metric could also have been interesting. However, the results proved to be very inconclusive, as can be seen in Fig. B.2.

3.3 Evaluation Dataset

Reproducibility, which is a primary concern in all sciences, is especially hard to guarantee in information retrieval (IR) systems. Ferro [4] discussed the main concerns of reproducibility

⁶ <https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.system/measure-nano-time.html>

of IR experiments: Even when the data and software is open source, often there are many hidden parameters and tunings that restrict the reproducibility. Secondly, the data collections used for experiments are often not publicly accessible. Lastly, during research, many aspects of the system get meta evaluated and tuned with inaccessible data.

The evaluation application developed during this thesis, together with a dataset of evaluation tasks, aims to resolve the previously mentioned concerns. The dataset upon which the evaluations were performed is the publicly available V3C1 [20] dataset also employed during the recent Video Browser Showdown competitions.

The evaluation dataset consisted of twenty-five known-item search tasks. For each task, four query components were developed. The query components were then combinatorially combined into fifteen evaluation queries. These queries were furthermore extended by different combinations of time differences. In total, every known-item search task was represented by 109 queries. The different time distances were employed to ensure no structural advantage for one or another algorithm for having “guessed” the time distances optimally.

Listing 3.1: Example specification with one task.

```

1  {
2      "tasks": [
3          {
4              "taskDescription": "Task on an election official in
5                  the US",
6              "taskType": "KIST",
7              "correctResult": {
8                  "V3C1ObjectId": "v_05057",
9                  "startAbs": 60.0,
10                 "endAbs": 80.0
11             },
12             "queries": [
13                 {
14                     "containers": [
15                         {
16                             "terms": [
17                                 {
18                                     "category": "text",
19                                     "type": [
20                                         "OCR"
21                                     ],
22                                     "data": "Manager"
23                                 }
24                             ]
25                         }
26                     ]
27                 }
28             ]
29         }
30     ]
31 }

```

```

27         ],
28         "config": {
29             "queryId": "3584f2e1-e562-4e4c-bf25-809c9f110100",
30             "hints": []
31         },
32         "timeDistances": []
33     }
34 ]
35 }

```

Each task, as can be seen in Listing 3.1, consists of a task description, a task type that can become relevant in an interactive evaluation session, the correct result, queries consisting of staged components which include the query terms, a configuration, and temporal distances between the queries.

The tasks are then combined into a specification file Listing A.1 that can be shared and should enable other developers to reproduce the tasks evaluated with the scoring algorithms developed during the scope of this thesis.

The evaluated tasks were inspired by the online interactive evaluation between SOMHunter and vitivr [21]. Furthermore, it should be noted that some queries would inherently render bad results due to being poorly formulated or due to the underlying retrieval models not retrieving a result.

3.4 Temporal Efficiency

Firstly, the temporal efficiency evaluation of the temporal scoring algorithms concerning execution time will be discussed. As can be seen in Fig. 3.2, the performances vary. However, further data is needed to come to a more conclusive result. Generally, it can be said that the top row of algorithms, Simple Temporal Scoring (STA), Sequential Temporal Scoring (SQA), vitivr Temporal Scoring (VITRIVR), and A* Temporal Scoring (ASTAR) tend to perform worse concerning execution time. While the bottom row of Normal Temporal Scoring (NA), Inverse Decay Temporal Scoring (IDA), Log-Normal Temporal Scoring (LNA), and Cluster Temporal Scoring (CLUSTER) have very similar performances with respect to execution time. The algorithms have been introduced in Section 2.3.

During the evaluation, each task was run individually, and the results of independent runs were counted individually. In a further evaluation, it could be interesting to run the tasks multiple times and then measure the mean and standard deviation on these results for the evaluation.

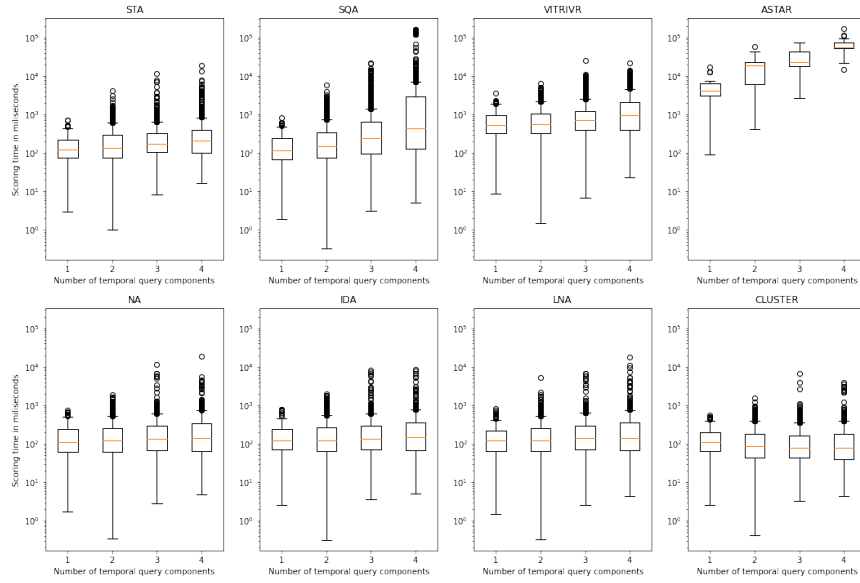


Figure 3.2: Comparing the performance of all temporal scoring algorithms with respect to execution time.

Firstly, it should be noted that ASTAR is so inefficient when including it in the following graphs, it makes the results of the other algorithms unreadable, even with a logarithmic scale.

For the following analysis, it is important to add that only the positive standard deviation is observed as the worst-case performance of the algorithm is only of interest and not necessarily best-case behaviour.

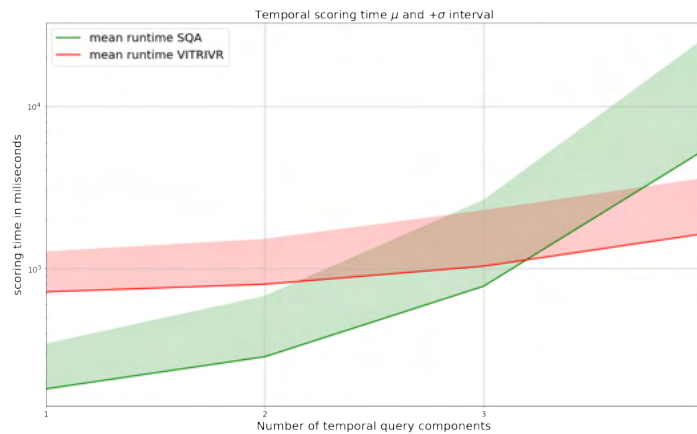


Figure 3.3: Temporal scoring time μ and $+\sigma$ interval for SQA and VITRVR.

The two slowest performing temporal algorithms will now be analysed concerning execution time apart from ASTAR as seen in Fig. 3.3. It can be observed that they behave very similarly, with a slight tendency of VITRIVR being more consistent, while SQA seems to be behaving in a somewhat exponential fashion. This behaviour is also in line with further investigations while creating the evaluations that SQA will run out of memory when confronted with too many query containers. Exponential time and memory complexity is not the desired behaviour of an algorithm in a productive system. In the future, it might be interesting to implement SQA with more favourable data types and introduce cut-offs for values below a threshold to reduce the memory load.

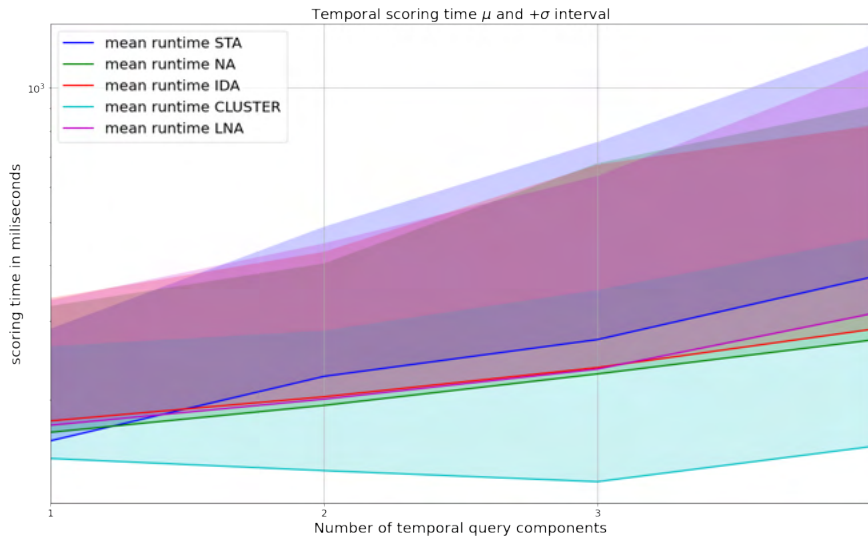


Figure 3.4: Temporal scoring time μ and $+\sigma$ interval for the top five fastest algorithms with respect to execution time.

The rest of the scoring algorithms can be analysed similarly, and the results are not entirely conclusive. As can be seen in Fig. 3.4, the scoring times of the algorithms all behave similarly and reasonably on the same level. This conclusion is an excellent result because they are all fast enough compared with the current implementation, and subsequently, the focus can be laid on the following evaluations.

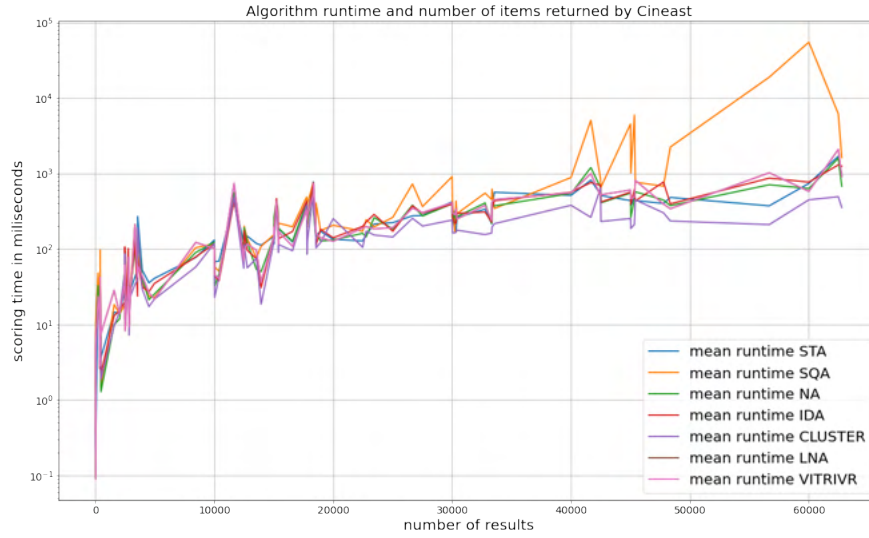


Figure 3.5: Temporal scoring time with respect to the number of retrieved results by Cineast.

While it is interesting to see the behaviour of the algorithms concerning the number of temporal query containers, it is also interesting to see if the number of results retrieved by Cineast has any influence upon the algorithms utilised. For this, the maximal number of results that Cineast would return for a query was modified to 15'000. This approach resulted in sometimes more than 65'000 results of temporal queries to be scored by the algorithms. The intuition would be that more results would change the execution time of the algorithms. As can be seen in Fig. 3.5 a trend can certainly be seen from all evaluated algorithms that would support this intuition; at the same time, it can also be seen that the differences in execution times do not change exponentially, which is good as it would allow us to use more retrieved results for scoring.

A more detailed picture of the six best performing algorithms with respect to execution time can now be analysed. The six best-performing algorithms, compared to the results of Fig. 3.2, are STA, SQA, CLUSTER, LNA, IDA, NA. The mean execution time as a line plot was added to have another comparative indicator.

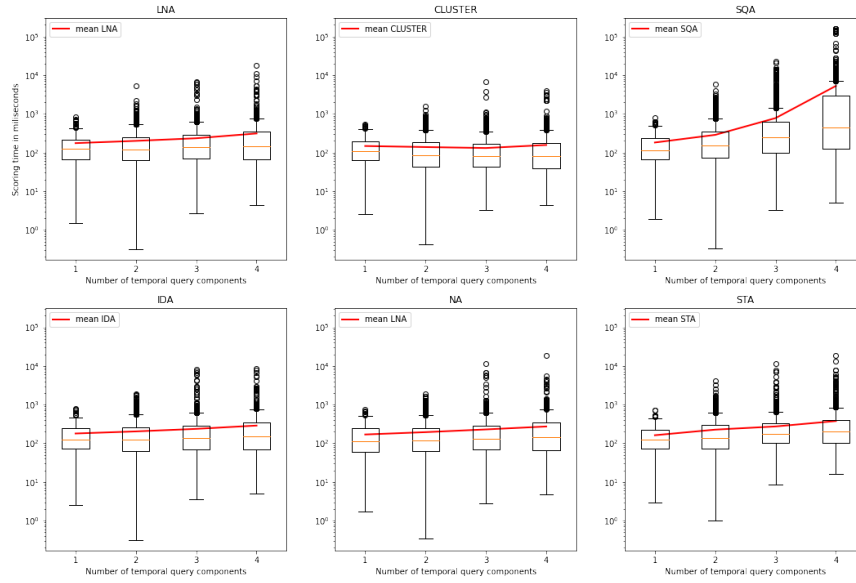


Figure 3.6: Comparing the boxplots of the six best performing scoring algorithms with respect to runtime.

Fig. 3.6 shows that the performance of all six of the algorithms is very similar, except SQA. SQA seems to suffer from an exponential increase in execution time. Otherwise, the speed concerning execution time depends on the query, so there is no single fastest algorithm.

3.5 Qualitative Performance

The evaluation of the qualitative performance of the temporal scoring algorithms is commenced by comparing the distributions of the search item positions.

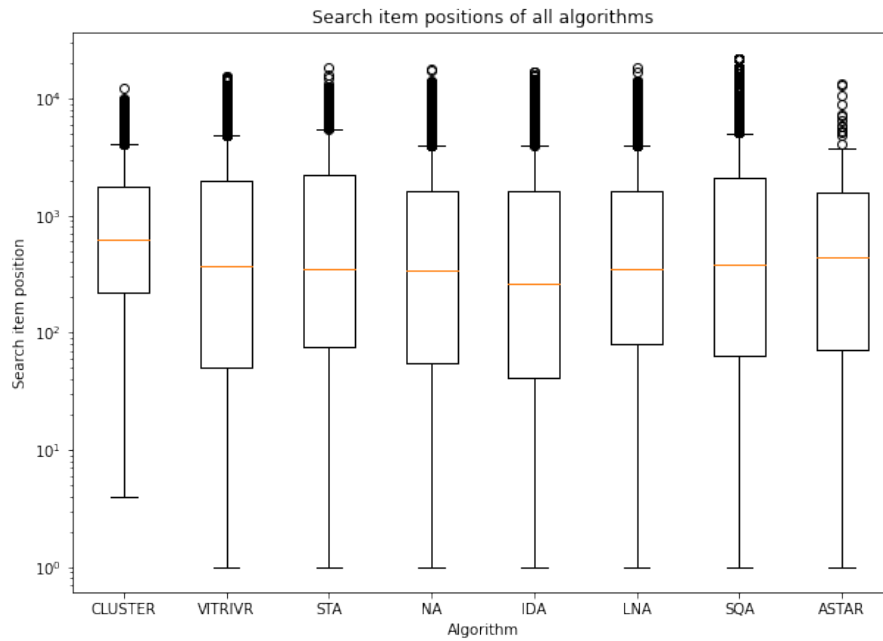


Figure 3.7: Searched item positions over all queries.

In Fig. 3.7, some specific algorithms can already be seen to tend to have a very favourable distribution regarding search item positions. As the discussion revolves around retrieval systems, search item positions anywhere above a certain threshold should not be considered as the user should not be expected to scroll 20'000 results to find the known item. Therefore the algorithms will be evaluated concerning some critical metrics, which can be observed in Table 3.1.

The cell with the best results from each column has been highlighted in blue. If ASTAR, which was determined as too slow, is best, the cells of ASTAR are additionally highlighted in yellow. The differences in whether an algorithm has found the searched item at all between the algorithms can vary because of the different ways the algorithms create temporal sequences. Additionally, it also shows that some query components did not retrieve the searched item at all. All tasks were kept because it may very well be that a user creates such a query, and the algorithms still have to be competitive under these circumstances.

Algorithm	Mean Search Item Position	Standard deviation Search Item Position	Finds result in %	Finds result in under 10'000 positions in %	Finds result in under 1'000 positions in %	Finds result in under 100 positions in %
<i>STA</i>	1720	2699	97.884	95.646	64.537	33.449
<i>SQA</i>	1960	3751	97.870	92.919	67.296	35.869
<i>NA</i>	1650	2811	97.869	94.906	68.500	36.107
<i>IDA</i>	1655	2979	97.870	94.584	69.878	42.089
<i>VITRIVR</i>	1736	2820	97.619	94.830	66.660	38.388
<i>CLUSTER</i>	1435	1872	97.870	97.829	59.302	12.132
<i>LNA</i>	1557	2599	97.869	95.558	68.770	30.540
<i>ASTAR</i>	1712	2855	97.980	94.949	67.677	36.364

Table 3.1: Performance comparison of the qualitative results of the algorithm evaluation. Blue is the best result apart from ASTAR, yellow signifies if ASTAR was better than any other algorithm.

While NA, LNA, and IDA are very similar from an algorithmic perspective, they sometimes render pretty different results. IDA proves to be reasonably reliable when the focus is on whether the algorithm finds the correct result below 1'000 and 100 results. At the same time, when looking at the mean search item position and the standard deviation, CLUSTER seems to have the lowest values, which would show that it is a relatively consistent algorithm. The values are, however, sometimes somewhat similar and can not show conclusive results. It is generally good to observe that multiple algorithms have better results than the baseline VITRIVR.

In Fig. 3.8 the results of the algorithms can be seen when only the distribution of the results between 1'000 and zero are considered. This comparison again shows that IDA performs very well with the most critical group of results and is better than the baseline.

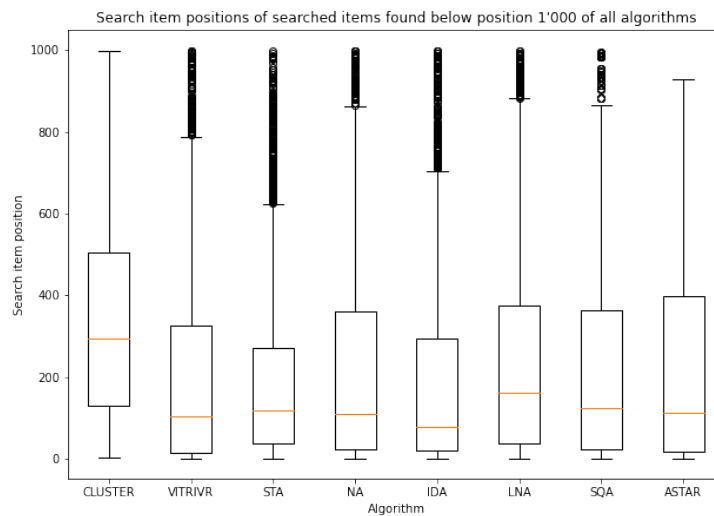


Figure 3.8: Searched item positions of results below 1'000.

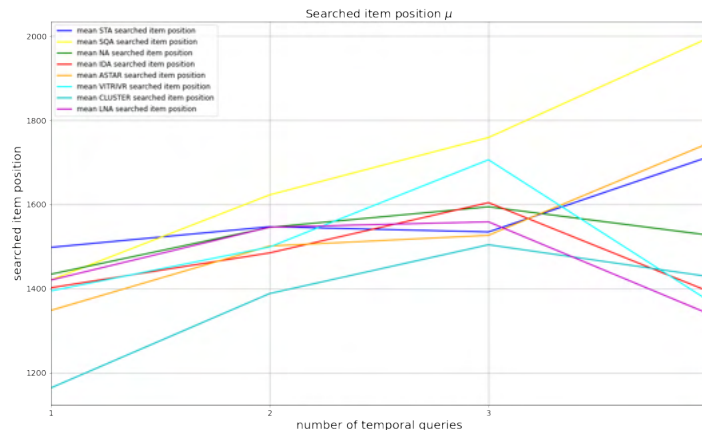


Figure 3.9: Search item position over number of queries per algorithm.

The average search item position over the number of temporal queries can be observed in Fig. 3.9. The figure shows that queries with more than three containers perform better than queries with precisely three, except SQA. It is important to note that the results from one query are not conclusive because single query requests often did not return any result at all, which is why they were excluded from this figure. The quality changes may have several reasons: estimating the timespans between the segments can significantly impact the results of an algorithm. At the same time, the results show that a time distance approach seems to be more sensible in most cases compared to a simple sequential search. Regarding the results of queries with multiple containers, it may also be hard to develop additional queries for the short segments provided by known item tasks.

Additionally, a graph to evaluate whether the search item position changed with the number of results retrieved by Cineast has been created. However, the results were inconclusive. These results will be excluded from the discussion. The graph is present in Fig. B.3.

3.6 Discussion

After presenting both the qualitative and the quantitative results of the evaluation, the results are discussed. Furthermore, the algorithms that would be good candidates to be implemented in the Cineast system are evaluated.

Firstly, there are two kinds of algorithms that have been evaluated. On one hand, various algorithms score the segments according to the correct sequence and the time distances between segments. On the other hand, there is SQA that only looks at the correct sequence. Before the temporal algorithms with time distances are assessed, SQA as described in Section 2.3.8 will be evaluated.

As seen in Fig. 3.3, SQA is not the fastest of the algorithms concerning execution time. However, SQA's runtime is not that much worse than the baseline runtime of VTRIVR. The implementation of SQA evaluated during the evaluation cycle is known to have some problems regarding the data types. It would be essential to make changes to the data

structures involved in a subsequential implementation to improve the runtime and remove the slightly exponential increase in runtime over several queries.

Due to its unique ability to produce similar results to the current system and the unique strength of not needing any temporal distance input, SQA, while attempting to implement the improvements mentioned beforehand, will be implemented in Cineast.

The algorithm evaluated to be the best performing in the category of temporal distance querying when implemented in Cineast would be IDA as described in Section 2.3.5.

As both seen in Fig. 3.4 and Fig. 3.6, IDA is one of the fastest performing algorithms. It is one of the most consistent and with similar variances compared to the other top contenders. The increase in runtime is also very steady and not exponential, which is an excellent property. Additionally, as seen in Fig. 3.5, the runtime does not drastically change when having to evaluate more results.

However, the more significant advantage of IDA comes with the qualitative evaluation of the algorithm. In Table 3.1 and the comparison in Fig. 3.8 the algorithm performs very well compared to the other top contenders and better than the baseline VITRIVR.

It is important to note that the differences between IDA, NDA, and NA are not substantial and any of these three have similar properties. In a further evaluation, it could be interesting to test out if different parameters of the functions employed in these three algorithms would produce different results.

To have an additional measure of quality combined with a temporal evaluation, a sort of VBS score function is introduced. This function is not representative of the scores evaluated at the VBS competition as a very low penalty for false submissions was employed because the automatic evaluation setup does not allow for any human interaction with the system. The formula employed during the evaluation is: $f_{KIS}^t(t, ws) = \lceil \max(0, 50 + 50 \cdot f_{TS}(t) - f_{WSP}(ws)) \rceil$ where t is the runtime of the algorithm, $f_{TS}(t) = \frac{T-t}{T}$ was the time penalty with $T = 5'000'000'000$, and $f_{WSP}(ws) = ws \cdot p$ the recall penalty with $p = 0.4$.

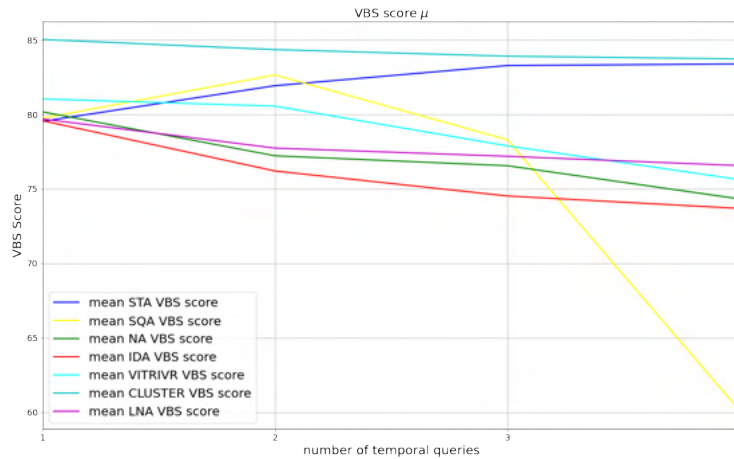


Figure 3.10: VBS score over number of queries per algorithm.

Fig. 3.10 again shows IDA perform similarly to the other contenders due to having good results early and a fast scoring time. Therefore, the evaluation is concluded with the verdict that IDA would be the best temporal distance algorithm to be implemented.

Lastly, the evaluation had shown that the chosen aggregation of results from the same segment within one temporal container had detriments when several features, especially semantic tags, were used within one or multiple containers. The issue with semantic tags is that the corresponding results have a binary score of zero or one. If now results from semantic tags were combined with results from, for example, Visual Text Co-Embedding, then the tag query results would dominate over the results from any other feature. It could be interesting to experiment with a different aggregation function within one container. The average score could be one such function to be evaluated.

4

Implementation

The two algorithms determined to be valuable concerning the evaluation metrics were implemented into the vitivr retrieval stack during this thesis. The implementation touched two systems, Cineast and vitivr-ng. In the following sections, the two parts of the implementation will be discussed. Due to the new different modalities of the temporal scoring algorithms, sequential and time distance, the corresponding query interfaces had to be implemented. Correspondingly, the temporal scoring view had to be updated. On the other hand, the temporal scoring algorithms were implemented in the Cineast back-end to be executed as part of a new query message handling.

4.1 Querying

The querying preferences of vitivr-ng were enhanced to enable the user with additional query parameters concerning the temporal querying. Additionally, new API messages to be exchanged between the front-end and the back-end were introduced. When querying, the user can define the temporal scoring mode between *sequential* where the user does not pass on any time distances between the query containers and *time distance* where the user will pass on time distances. Additionally, the user can set a maximal length of the sequence that, when restrictive enough, should help the user to retrieve the correct sequence. The changes in vitivr-ng were implemented in the *Angular*⁷ framework based on the programming language *TypeScript*⁸.

⁷ <https://angular.io/>

⁸ <https://www.typescriptlang.org/>

Temporal mode

temporal distance ▼

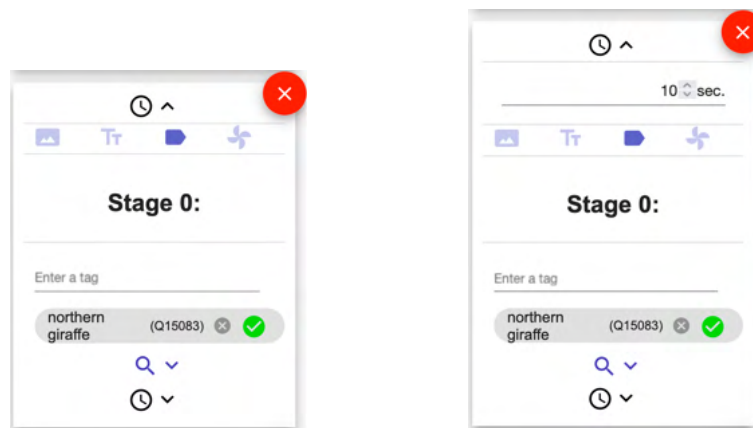
Max sequence length

600 sec.

Set

Figure 4.1: Temporal query preference settings in vitrivr-ng.

In Fig. 4.1, the aforementioned settings can be observed. The data will then be transmitted to the Cineast instance connected with the WebSocket interface.



(a) A query container in the sequential temporal query mode.

(b) A query container in the time distance temporal query mode.

Figure 4.2: Query containers in both temporal query modes.

Additionally, in Fig. 4.2 query, containers for a temporal query with three or more containers can be observed. In Fig. 4.2a the buttons to change the ordering are the only additional settings on the level of query containers, while in Fig. 4.2b the user can furthermore set the time distance to the previous container.

```

1 TemporalQueryV2 : Object
2 {
3   "queries": Array<TemporalQueryComponent>,
4   "config": QueryConfig,
5   "timeDistances": Array<Float>,
6   "maxLength": Float,
7   "messageType": "Q_TEMP"
8 }

```

Listing 4.1: TemporalQueryV2 object: Q_TEMP.

The additional information that should be transferred from the front-end to the back-end is the time distance array between the temporal segments and a max time for the whole segment. As seen in Listing 4.1, the query message for the new temporal query type was adapted. The new and adapted query message is backwards compatible as the existing *TemporalQuery* has been extended, and the new parameters are optional.

4.2 Result View

Additionally to an updated querying interface in the front-end, the result view was also updated and simplified. The results of the temporal scoring algorithms from the back-end will be transmitted with the newly introduced *TemporalQueryResult*. The result returned to the user will include the usual QR_START, QR_END, QR_SEGMENT, QR_OBJECT, QR_SIMILARITY and then additionally QR_TEMPORAL.

```

1 TemporalQueryResult : Object
2 {
3     "content": Array<TemporalObject>,
4     "count": Integer,
5     "queryId": String,
6     "messageType": "QR_TEMPORAL"
7 }
```

Listing 4.2: TemporalQueryResult object: QR_TEMPORAL.

The new TemporalQueryResult can be seen in Listing 4.2 where the front-end receives the information to extract and display the temporal sequences. Each temporal sequence consists of all temporal sequences created from the requested temporal query. No further information is required as the front-end already has all the other information from the other response messages. The score of the temporal objects will be used to order the content.

```

1 TemporalObject : Object
2 {
3     "segments": Array<String>,
4     "objectId": String,
5     "score": Float
6 }
```

Listing 4.3: TemporalObject object to transfer the segments contained in an object and the score.

In Listing 4.3, the TemporalObject that returns the max-pooled score of the temporal sequences of an object, as well as the segments that have been part of a temporal sequence, can be seen. The segments array contain unique segments ordered by their temporal position within the object.

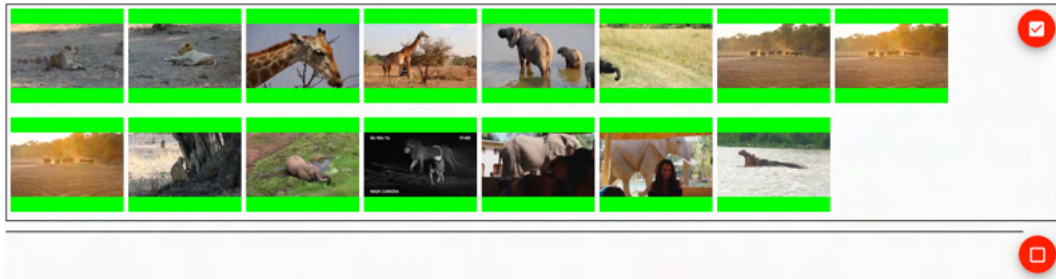


Figure 4.3: Result view of one expanded result object and one toggled object.

The display of the results can be seen in Fig. 4.3. Here result containers containing all the segments with results to the previously introduced query in Fig. 2.1 can be seen. The first container is expanded, while the second one is toggled. The toggle functionality has been introduced to enable a more accessible search experience as some objects may contain many segments that are not relevant to the search; those irrelevant objects, however, then clutter the view for the user.

4.3 Temporal Scoring

The temporal scoring part was implemented in the back-end of vitrivr, the retrieval engine Cineast. The implementation of Cineast is in the programming language Java. The scoring was transferred from the previous position of the front-end to the back-end. This change is to relieve the front-end from expensive calculations and do the calculations on the server-side of the retrieval stack. The front-end already has too many computations to execute. This relief is essential as most end users will not have the fastest hardware while most servers usually have a higher computational power. During the implementation, backwards compatibility was ensured by adding new independent classes instead of altering the existing code.

In Cineast, the temporal message then gets handled accordingly. Firstly, the information from all temporal queries is retrieved, and the sequence number and all the results concerning this particular sequence number will be stored. The staged similarity queries will still be applied as previously, with the first stage acting as a filter on all subsequent stages and only the results from the last stage being saved and returned. Sequential temporal scoring (SQA) introduced in Section 2.3.8 is applied if no time distances have been provided. Else, inverse decay temporal scoring (IDA) introduced in Section 2.3.5 is applied.

The basic idea is that users know intuitively how much time passes between the segments for some retrieval tasks, which is likely in, for example, a visual task. However, sometimes they do not know the distances between the segments but know the order, which is, for example, likely in a textual task. The max time is there to cap the results that are too large and to return a smaller subset of results instead of the whole package.

Internally in Cineast, not only the first 10'000 results of a query will be requested, however all results as this may enhance the chance of finding a good segment. Furthermore, the scoring will be applied on the prefiltered items if a staged query was executed beforehand. There is an additional duplicate check to remove duplicate segments from the result list to clean up redundant clutter. The scored sequences will then be max-pooled over the best

sequence and combined into result objects as can be observed in Section 2.2. These objects will then be returned to the user and displayed as discussed in Section 4.2.

5

Related Work

Other multimedia retrieval systems will be discussed in the following sections that have employed temporal querying in their retrieval interfaces and engines. The other systems participating in recent instalments of the Video Browser Showdown that enable temporal querying will be compared to the implementation of this thesis, and commonalities and differences will be evaluated.

5.1 VBS participants

The evaluation [18] of the most recent evaluated instalment of the VBS competitions in 2019 showed that frequent use of temporal queries increases the performance of retrieval systems. When combined with other potent querying modalities, the potential of temporal querying enabled successful teams to achieve good results in the expert evaluation. Moreover, while the systems rarely employed temporal querying during the previous instalments of the VBS competition [12], multiple systems have employed such query specifications during the last VBS competition in 2020.

5.1.1 Viret

In the Viret system, as described in the papers for the 27th ACM International Conference on Multimedia [14] [13] [11], temporal aggregation is evaluated for each modality separately. If the user does not provide the second query for a given modality, the system considers defaulting relevance score values. Viret currently supports a sequence of two queries for the same modality.

Viret also introduces the notion of a primary/secondary query. If the first query is set as primary, the overall score of a frame o_i for a given modality M is computed from $score_{q_1}^M[i]$ and $max_{j=i+1}^{i+k}(score_{q_2}^M[j])$, where $score_{q_i}^M$ represents an array of relevance scores with respect to q_i for a temporally ordered sequence of frames. Where k is, similar to the vitrivr implementation, the configurable threshold.

The scoring system calculates the overall score of the combined query, $score_{q_i}^{K_1, K_2}$ by multiplication for textual search, addition for colour-based and query-by-example models, and

logical AND for localized object models.

Viret was one of the first systems to employ temporal querying during the 2019 VBS competition successfully. However, their system is limited to a sequence of two queries and thus limited compared to the vitivr implementation. Furthermore, the Viret system only permits temporal queries of the same modalities due to the different score combination methodologies. While their system has shown to perform well, vitivr system's differences are significant enough to distinguish vitivr positively.

5.1.2 Verge

The multimodal aggregation module of Verge [2] fuses the results of two or more search modules. In order to perform a temporal search, a query using multiple features of two adjacent shots is received, the system retrieves the top-N relevant images for one of the queries, and finally, this list is re-ranked by considering the features of the adjacent shot.

The Verge group uses a multimodal aggregation approach, which combines graph-based and non-linear aggregation. The temporal search works similar to the scoring introduced by the Viret team, with having a primary and a secondary shot.

Verge has employed their version of the temporal queries during the 2020 instalment of the VBS competition. Their algorithm is similar to the one employed by Viret and has the same drawbacks concerning the number of potential temporal queries. The positive thing to note is that they employ their temporal scoring after the score adjustments of multimodal queries, similar to the implementation in vitivr.

5.1.3 Vireo

Vireo employed their implementation of temporal scoring for the VBS 2020 competition [15]. The Vireo team implemented their very own and unique version of temporal scoring. The Vireo system displays two canvases to the user to input two object-sketch queries at the timestamp t and t' with $t < t'$. The similarity of a video $V = O_1, \dots, O_k$ and two queries $Q_t, Q_{t'}$. Is calculated as follows:

1. Calculate two sets of similarities $Sim(Q_t, O_1), \dots, Sim(Q_t, O_k)$ and $Sim(Q_{t'}, O_1), \dots, Sim(Q_{t'}, O_k)$
2. Construct array $MaxSim(Q_t, V) = [s_1^t, \dots, s_k^t]$ with $s_1^t = Sim(Q_t, O_1)$ and $s_i^t = \max(s_{i-1}^t, Sim(Q_t, O_i))$ for $i > 1$. Here the order from 1 to k represents the temporal order of the key-frames. This is an increasing array where an element s_i^t represents the maximum similarity of the query Q_t to the video segment starting from the first key-frame to the key-frame i.
3. Construct array $MaxSim(Q_{t'}, V) = [s_1^{t'}, \dots, s_k^{t'}]$ with $s_k^{t'} = Sim(Q_{t'}, O_k)$ and $s_i^{t'} = \max(s_{i+1}^{t'}, Sim(Q_{t'}, O_i))$ for $i < k$. Here the order from 1 to k represents the temporal order of the key-frames. This is a decreasing array where an element $s_i^{t'}$ of this array represents the maximum similarity of the query $Q_{t'}$ to the video segment starting from the key-frame i to the last key-frame.

4. Calculate $Sim((Q_t, Q_{t'}), V) = \max_{i=1}^{k-1} (s_i^t + s_{i+1}^{t'})$.

This approach enables the user to remember only the order $t < t'$ rather than the exact interval between two queries. The implementation proposed by this paper thus also features a sequence-based query formulation similar to the implementation of sequential temporal scoring as discussed in Section 2.3.8. The drawback of Vireos implementation is that they only support two adjacent queries compared to multiple in the current vitivr implementation.

5.1.4 SOM-Hunter V2

SOM-Hunter first participated in the VBS competition in 2020 [9] and showed an up-and-coming competitor winning by solving 15 out of 22 known-item search tasks. The SOM-Hunter team has now improved their system for VBS 2021 [24]. SOM-Hunter introduced temporal queries for their first system in 2020.

Similarly to the other systems discussed, they have introduced three adjacent temporal queries and limit their users to this number of queries. As the evaluation of this thesis has shown, this might be a good number of temporal queries for the VBS competition. However, for longer videos, it might be beneficial to introduce the possibility of more temporal queries. Their main limitation seems to be the structure of their user interface. The temporal query containers provided by vitivr prove substantial help for the users to make the temporal querying clear and easy to understand.

5.1.5 W2VV++ BERT

The W2VV++ model BoW variant was recently successfully integrated into both Viret and SOM-Hunter. The Siret team now proposed a W2VV++ model [16] based on the more complex BERT variant. The novel and interesting feature of the proposed system is the context-aware query ranker, where individual W2VV++ query results are being re-ranked. It relaxes the temporal ordering to a temporal co-location context. The system makes the following assumptions for their implementation:

1. Users can describe the seen scenes in much more detail when asked.
2. The user descriptions span over one scene, and no single segment can contain all the information provided by the user.
3. All subqueries correspond to a short time window of the searched scene.
4. There are not too many files that correspond to a co-location context query in the dataset
5. Users often describe longer-lasting events

The system proposed by the Siret team employs a sliding window approach where they aggregate the scores of the relevance of the results within the step size by either product or sum aggregations. While this proposal tries to satisfy the assumptions suggested, it is significantly different from the assumptions of the vitivr implementation as described in

Section 2.2. The cluster temporal scoring as described in Section 2.3.4 employed a similar approach, where the algorithm constructed clusters with temporally close results. However, the evaluation has shown the results to be quite bad in comparison with the other systems.

5.1.6 Visione

Similar to other systems presented here, Visione also introduced temporal querying for their 2021 version of their retrieval system [1]. They employ a similar strategy to the Viret system with the possibility to provide two queries that should occur within a certain threshold. Their system, however, has not added any additional new functionality, and the usefulness compared to the other systems will have to prove itself during the 2021 instalment of the competition.

5.2 Other Scoring/Rank Aggregation Algorithms

During the investigations of this thesis into possible algorithms, many other potential approaches to the problem described in Section 2.2 have been encountered. In the following subsections, some additional ideas encountered during the investigations will be discussed.

5.2.1 Prize-Collecting Steiner Tree Problem

During the exploration of potential algorithms, graph covering problems were also evaluated. The Prize-Collecting Steiner Tree Problem as described in [5] is a graph problem with a set of edges in an edge- and node-weighted graph chosen to satisfy some covering constraint. The problem asks for a subtree that minimizes the total cost of all edges in the subtree plus the total profit of all vertices not contained in the subtree. The Prize-Collecting Steiner Tree graph would consist of segments as nodes and distances between the different segments from the different containers as edges in the exploration. The problem will have to be changed from a minimization problem to a maximization problem. However, this should be possible. Together with the algorithmic framework described in [10] it could be interesting to implement such an algorithm based on the Prize-Collecting Steiner Tree Problem.

5.2.2 Maximum Rank Aggregation Problems

As already discussed in Section 2.2 the problem of temporal scoring in a Multimedia retrieval system is also often seen as a rank aggregation problem. The goal of temporal scoring is to find the best sequence of segments from multiple result sets. According to [3], a maximum rank aggregation problem is given by a set of m permutations of a set of size n , and the goal is to find a consensus permutation with minimum distance to the given permutation. The maximum rank aggregation problem is NP-hard. In the case of this thesis, the given permutation could be a temporal sequence with a perfect score and an element in each temporal container and the perfect time distances.

A possible implementation of a maximum rank algorithm on the problem of this thesis would create, for each object, potential permutations and evaluate the distances between the

permutations and the given permutation of a perfect result. For each object, the permutation with the closest distance would determine the score of this object.

The algorithm discussed in [3] could be a start on how to calculate the maximum rank aggregation problem. However, the creation of potential permutations and the calculation of Maximum Ranking are very time-consuming and thus very likely not a tractable solution.

5.2.3 Weighted Minimum Feedback Arc Set Problem in Tournaments

In the *weighted minimum feedback arc set problem in tournaments* as described in [23] a set of elements V is given, nonnegative weights $w_{i,j}$ and w_j , for each pair of distinct elements i and j , and a permutation π that minimizes the weights of pairs of elements out of order concerning the permutation is wanted. In the problem of this thesis, the input is a collection of permutations of segments of one object, and $w_{i,j}$ is the fraction of orderings in which i is ordered before j . The problem would be further constrained that all output permutations Π must be consistent with a partial order P . This problem is, same as the maximum rank aggregation problem, NP-hard. There do exist deterministic algorithms that could be an exciting approach to temporal scoring.

6

Conclusion

This chapter wraps up the thesis by reflecting on the contributions and suggesting possible future work.

6.1 Conclusion

During this thesis, new algorithms for temporal scoring were developed and evaluated. For the evaluation with regards to quality and time effectiveness, a stand-alone evaluation application has been developed. A task specification file format to define tasks for the evaluation application to evaluate the algorithms was established. The tasks were manually defined and inspired by earlier known-item search tasks.

Based on the results, two algorithms were further developed and implemented in Cineast. This means that vitivr's temporal scoring was moved from the front-end of vitivr-ng to the retrieval engine Cineast. New API endpoints to facilitate this change were created, and the handling of the temporal query messages was extended. Three additional query parameters were introduced to communicate the necessary information from the front-end to the back-end for the temporal rank aggregation. The results were then deployed and successfully employed during the 10th anniversary of the Video Browser Showdown.

6.2 Further Work

With a focus on the Lifelog Search Challenge, the evaluation of the following ideas might be necessary.

- Additional query parameters could be enabled for the temporal queries to be passed on to the back-end with, for example, a setting in which a user can define which query containers must be necessarily present in the sequences and others which are optional.
- The display of the temporal sequences could be further enhanced by colour-coding the query containers and the corresponding segments to clarify which segment has contributed to which query and establish the order of temporal sequences further.

- A different result score aggregation function could be used for the result aggregation within one container to weaken the effect of binary scored features.

Furthermore, the following broader questions could be interesting for further work:

- The evaluation specification could be extended with additional parameters for the algorithms to allow future changes.
- Additional temporal scoring algorithms could be developed and evaluated. Some ideas for other algorithms were discussed in Chapter 5.
- Additional analysis tools in the front-end could be implemented for the users to understand their queries' results better.
- The temporal querying interface in vitivr-ng could be remodelled to enable a more compact search experience
- All scoring algorithms introduced implement adding up the individual results for segments. Other ways of combining the result scores from the queries could be tried out in a future iteration. Subsequently, additional evaluation tasks with more mixed modalities could be developed to test different score combination methodologies.

Bibliography

- [1] Giuseppe Amato, Paolo Bolettieri, Fabrizio Falchi, Claudio Gennaro, Nicola Messina, Lucia Vadicamo, and Claudio Vairo. Visione at video browser showdown 2021. In Jakub Lokoč, Tomáš Skopal, Klaus Schoeffmann, Vasileios Mezaris, Xirong Li, Stefanos Vrochidis, and Ioannis Patras, editors, *MultiMedia Modeling*, pages 473–478, Cham, 2021. Springer International Publishing. ISBN 978-3-030-67835-7.
- [2] Stelios Andreadis, Anastasia Moutzidou, Konstantinos Apostolidis, Konstantinos Gkoutakos, Damianos Galanopoulos, Emmanouil Michail, Ilias Gialampoukidis, Stefanos Vrochidis, Vasileios Mezaris, and Ioannis Kompatsiaris. Verge in vbs 2020. In Yong Man Ro, Wen-Huang Cheng, Junmo Kim, Wei-Ta Chu, Peng Cui, Jung-Woo Choi, Min-Chun Hu, and Wesley De Neve, editors, *MultiMedia Modeling*, pages 778–783, Cham, 2020. Springer International Publishing. ISBN 978-3-030-37734-2.
- [3] Christian Bachmaier, Franz Josef Brandenburg, Andreas Gleißner, and Andreas Hofmeier. On maximum rank aggregation problems. In Thierry Lecroq and Laurent Mouchard, editors, *Combinatorial Algorithms*, pages 14–27, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45278-9.
- [4] Nicola Ferro. Reproducibility challenges in information retrieval evaluation. *J. Data and Information Quality*, 8(2), January 2017. ISSN 1936-1955. doi: 10.1145/3020206.
- [5] Takuro Fukunaga. Covering problems in edge- and node-weighted graphs. *Discrete Optimization*, 20:40–61, 2016. ISSN 1572-5286. doi: <https://doi.org/10.1016/j.disopt.2016.03.001>.
- [6] Ralph Gasser, Luca Rossetto, and Heiko Schuldt. Multimodal multimedia retrieval with vitivr. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, ICMR '19, page 391–394, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367653. doi: 10.1145/3323873.3326921.
- [7] Ralph Gasser, Luca Rossetto, Silvan Heller, and Heiko Schuldt. Cottontail db: An open source database system for multimedia retrieval and analysis. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 4465–4468, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379885. doi: 10.1145/3394171.3414538.
- [8] S. Heller, L. Sauter, H. Schuldt, and L. Rossetto. Multi-stage queries and temporal scoring in vitivr. In *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–5, 2020. doi: 10.1109/ICMEW46912.2020.9105954.

- [9] Miroslav Kratochvíl, Patrik Veselý, František Mejzlík, and Jakub Lokoč. Som-hunter: Video browsing with relevance-to-som feedback loop. In Yong Man Ro, Wen-Huang Cheng, Junmo Kim, Wei-Ta Chu, Peng Cui, Jung-Woo Choi, Min-Chun Hu, and Wesley De Neve, editors, *MultiMedia Modeling*, pages 790–795, Cham, 2020. Springer International Publishing. ISBN 978-3-030-37734-2.
- [10] Ivana Ljubić, René Weiskircher, Ulrich Pfersch, Gunnar W. Klau, Petra Mutzel, and Matteo Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical Programming*, 105(2):427–449, Feb 2006. ISSN 1436-4646. doi: 10.1007/s10107-005-0660-x.
- [11] Jakub Lokoč, Gregor Kovalčík, and Tomáš Souček. Viret at video browser showdown 2020. In Yong Man Ro, Wen-Huang Cheng, Junmo Kim, Wei-Ta Chu, Peng Cui, Jung-Woo Choi, Min-Chun Hu, and Wesley De Neve, editors, *MultiMedia Modeling*, pages 784–789, Cham, 2020. Springer International Publishing. ISBN 978-3-030-37734-2.
- [12] Jakub Lokoč, Gregor Kovalčík, Bernd Münzer, Klaus Schöffmann, Werner Bailer, Ralph Gasser, Stefanos Vrochidis, Phuong Anh Nguyen, Sitapa Rujikietgumjorn, and Kai Uwe Barthel. Interactive search or sequential browsing? a detailed analysis of the video browser showdown 2018. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(1), February 2019. ISSN 1551-6857. doi: 10.1145/3295663.
- [13] 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*, ICMR '19, page 177–181, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367653. doi: 10.1145/3323873.3325034.
- [14] Jakub Lokoč, Gregor Kovalčík, 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*, MM '19, page 1777–1785, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368896. doi: 10.1145/3343031.3351046.
- [15] Phuong Anh Nguyen, Jiaxin Wu, Chong-Wah Ngo, Danny Francis, and Benoit Huet. Vireo @ video browser showdown 2020. In Yong Man Ro, Wen-Huang Cheng, Junmo Kim, Wei-Ta Chu, Peng Cui, Jung-Woo Choi, Min-Chun Hu, and Wesley De Neve, editors, *MultiMedia Modeling*, pages 772–777, Cham, 2020. Springer International Publishing. ISBN 978-3-030-37734-2.
- [16] Ladislav Peška, Gregor Kovalčík, Tomáš Souček, Vít Škrhák, and Jakub Lokoč. W2vv++ bert model at vbs 2021. In Jakub Lokoč, Tomáš Skopal, Klaus Schoeffmann, Vasileios Mezaris, Xirong Li, Stefanos Vrochidis, and Ioannis Patras, editors, *MultiMedia Modeling*, pages 467–472, Cham, 2021. Springer International Publishing. ISBN 978-3-030-67835-7.

- [17] L. Rossetto, I. Giangreco, and H. Schuldt. Cineast: A multi-feature sketch-based video retrieval engine. In *2014 IEEE International Symposium on Multimedia*, pages 18–23, 2014. doi: 10.1109/ISM.2014.38.
- [18] L. Rossetto, R. Gasser, J. Lokoč, W. Bailer, K. Schoeffmann, B. Muenzer, T. Souček, 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*, 23:243–256, 2021. doi: 10.1109/TMM.2020.2980944.
- [19] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. Vitriivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM ’16, page 1183–1186, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450336031. doi: 10.1145/2964284.2973797.
- [20] Luca Rossetto, Heiko Schuldt, George Awad, and Asad A. Butt. V3c – a research video collection. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling*, pages 349–360, Cham, 2019. Springer International Publishing. ISBN 978-3-030-05710-7.
- [21] Luca Rossetto, Ralph Gasser, Silvan Heller, Mahnaz Parian-Scherb, Loris Sauter, Florian Spiess, Heiko Schuldt, Ladislav Peska, Tomas Soucek, Miroslav Kratochvil, Frantisek Mejzlik, Patrik Vesely, and Jakub Lokoc. On the user-centric comparative remote evaluation of interactive video search systems. *IEEE MultiMedia*, pages 1–1, 2021. doi: 10.1109/MMUL.2021.3066779.
- [22] Klaus Schoeffmann. Video browser showdown 2012-2019: A review. In *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, pages 1–4, 2019. doi: 10.1109/CBBI.2019.8877397.
- [23] Anke van Zuylen and David P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In Christos Kaklamanis and Martin Skutella, editors, *Approximation and Online Algorithms*, pages 260–273, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-77918-6.
- [24] Patrik Veselý, František Mejzlík, and Jakub Lokoč. Somhunter v2 at video browser showdown 2021. In Jakub Lokoč, Tomáš Skopal, Klaus Schoeffmann, Vasileios Mezaris, Xirong Li, Stefanos Vrochidis, and Ioannis Patras, editors, *MultiMedia Modeling*, pages 461–466, Cham, 2021. Springer International Publishing. ISBN 978-3-030-67835-7.



Task Specification

```
1 {
2   "$schema": "http://json-schema.org/schema#",
3   "title": "Specification",
4   "type": "object",
5   "properties": {
6     "tasks": {
7       "type": "array",
8       "items": {
9         "$ref": "#/$defs/task"
10      }
11    }
12  },
13  "required": [
14    "tasks"
15  ],
16  "$defs": {
17    "task": {
18      "type": "object",
19      "properties": {
20        "taskDescription": {
21          "type": "string",
22          "description": "Task description(KIS-T) / information
23                        about the video sequence (KIS-V) "
24        },
25        "taskType": {
26          "type": "string",
27          "enum": [
28            "KIST",
29            "KISV"
30          ]
31        }
32      }
33    }
34  }
35 }
```

```
30         "description": "The type of the task at hand for better
31             classification"
32     },
33     "correctResult": {
34         "$ref": "#/$defs/correctResult"
35     },
36     "queries": {
37         "$ref": "#/$defs/Query",
38         "description": "Queries used to retrieve the data. One
39             temporal query consists of multiple queries done in
40             succession."
41     },
42     "timeDistances": {
43         "type": "array",
44         "items": {
45             "type": "number"
46         },
47         "description": "List of times between the queries used
48             for the temporal scoring of closeness."
49     }
50 },
51 "required": [
52     "taskDescription",
53     "taskType",
54     "correctResult",
55     "queries"
56 ],
57 "correctResult": {
58     "type": "object",
59     "properties": {
60         "V3C1ObjectId": {
61             "type": "string",
62             "description": "V3C1 object ID to locate the media item.
63                 ObjectId is enough because with the timestamps we
64                 can easily evaluate if a returned result is within
65                 those boundaries."
66         },
67         "start": {
68             "type": "number",
69             "description": "Start of the correct result"
70         },
71         "end": {
```

```
66         "type": "number",
67         "description": "End of the correct result"
68     }
69 },
70     "required": [
71         "V3C1ObjectId",
72         "start",
73         "end"
74     ]
75 },
76     "Query": {
77         "type": "object",
78         "properties": {
79             "containers": {
80                 "type": "array",
81                 "items": {
82                     "$ref": "#/$defs/QueryContainer"
83                 },
84                 "description": "List of QueryContainer"
85             },
86             "config": {
87                 "$ref": "#/$defs/QueryConfig"
88             }
89         },
90         "required": [
91             "containers",
92             "config"
93         ]
94     },
95     "QueryContainer": {
96         "type": "object",
97         "properties": {
98             "terms": {
99                 "type": "array",
100                 "items": {
101                     "$ref": "#/$defs/Term"
102                 },
103                 "description": "List of Terms"
104             }
105         },
106         "required": [
107             "terms"
108         ]
109     }
```



```
109     },
110     "Term": {
111       "type": "object",
112       "properties": {
113         "category": {
114           "type": "string",
115           "enum": [
116             "text",
117             "image",
118             "sketch",
119             "filter"
120           ],
121           "description": "Collection of categories to employ while
                           searching"
122         },
123         "type": {
124           "type": "array",
125           "items": {
126             "type": "string",
127             "enum": [
128               "metadata",
129               "OCR",
130               "ASR",
131               "concept",
132               "localizedObject",
133               "caption",
134               "jointEmbedding",
135               "custom",
136               "globalFeatures",
137               "localFeatures",
138               "feedbackModel",
139               "color",
140               "edge",
141               "motion",
142               "semanticSegmentation",
143               "B/W",
144               "dominantColor",
145               "resolution",
146               "numberOfObjects"
147             ]
148           },
149           "description": "Collection of categories to employ while
                           searching"
```

```
150     }
151 },
152 "allOf": [
153   {
154     "if": {
155       "properties": {
156         "category": {
157           "const": "text"
158         }
159       }
160     },
161     "then": {
162       "if": {
163         "properties": {
164           "type": {
165             "const": "concept"
166           }
167         }
168       },
169       "then": {
170         "properties": {
171           "data": {
172             "type": "string",
173             "description": "The query data which is in the
                           locally normal json format as base64 string
                           with the format data:application/json;base64
                           ,data."
174           }
175         }
176       },
177       "else": {
178         "properties": {
179           "data": {
180             "type": "string",
181             "description": "The query data to be queried for
                           for example ASR or OCR as a plain string."
182           }
183         }
184       }
185     }
186   },
187   {
188     "if": {
```

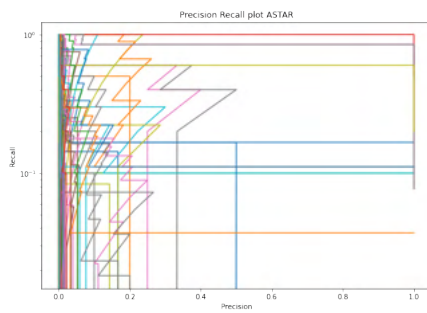
```
189         "properties": {
190             "category": {
191                 "const": "image"
192             }
193         },
194     },
195     "then": {
196         "properties": {
197             "data": {
198                 "type": "string",
199                 "description": "The query data as base64 string
200                             with the format data:image/{file_format};base6
201                             4,data"
202             }
203         }
204     },
205     {
206         "if": {
207             "properties": {
208                 "category": {
209                     "const": "sketch"
210                 }
211             },
212             "then": {
213                 "properties": {
214                     "data": {
215                         "type": "string",
216                         "description": "The query data as base64 string
217                             with the format data:image/{file_format};base6
218                             4,data"
219                     }
220                 }
221             },
222             {
223                 "if": {
224                     "properties": {
225                         "category": {
226                             "const": "filter"
227                         }
228                     }
229                 }
230             }
231         }
232     }
233 }
```

```
228     },
229     "then": {
230       "properties": {
231         "data": {
232           "type": "string",
233           "description": "The query data which is in the
                           locally normal json format as base64 string
                           with the format data:application/json;base64,
                           data."
234         }
235       }
236     }
237   },
238 ],
239 "required": [
240   "type",
241   "data"
242 ]
243 },
244 "QueryConfig": {
245   "type": "object",
246   "properties": {
247     "queryId": {
248       "type": "string",
249       "description": "Same as what was in the request if
                       specified or else will be randomly generated"
250     }
251   },
252   "required": []
253 }
254 }
255 }
```

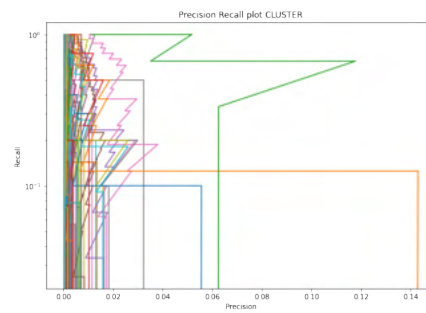
Listing A.1: Task specification for temporal scoring algorithm evaluation

B

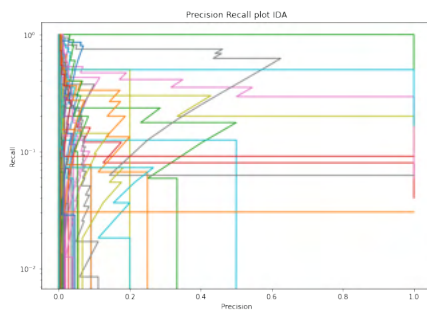
Evaluation Plots



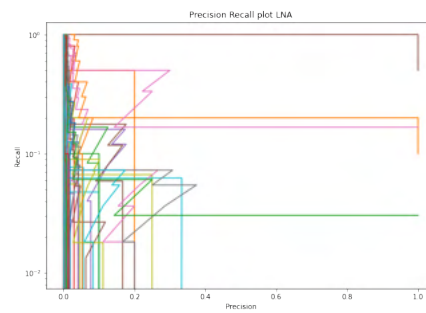
(a) Precision-Recall plot of ASTAR



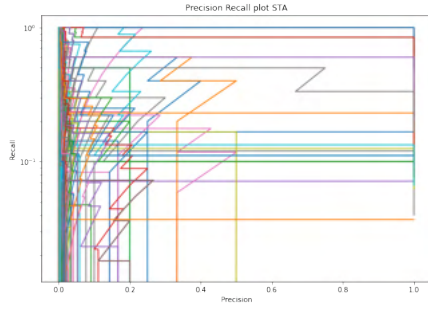
(b) Precision-Recall plot of CLUSTER



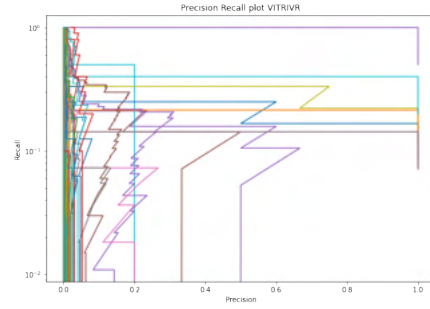
(c) Precision-Recall plot of IDA



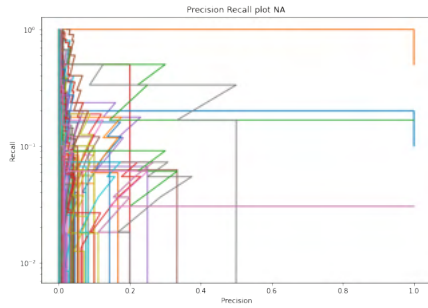
(d) Precision-Recall plot of LNA



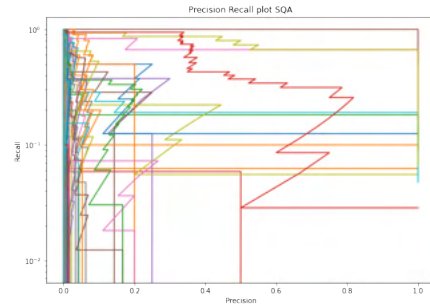
(a) Precision-Recall plot of STA



(b) Precision-Recall plot of VITRIVR



(c) Precision-Recall plot of NA



(d) Precision-Recall plot of SQA

Figure B.2: Precision Recall plots of all evaluated algorithms

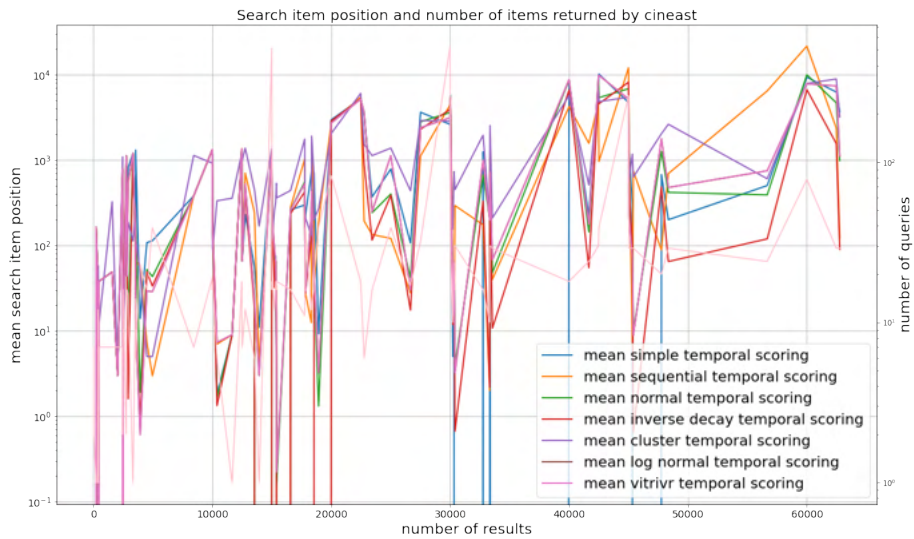


Figure B.3: Search item position compared with retrieval results from Cineast

Declaration on Scientific Integrity

Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor

Viktor Gsteiger

Matriculation number — Matrikelnummer

2018-054-700

Title of work — Titel der Arbeit

Evaluating Algorithms for Temporal Queries in Ad-Hoc Video Retrieval

Type of work — Typ der Arbeit


Bachelor thesis

Declaration — Erklärung

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 30.06.2021



Signature — Unterschrift