Int J Multimed Info Retr
DOI 10.1007/s13735-017-0122-2

REGULAR PAPER

Multicontext-adaptive indexing and search for large-scale video navigation

Diep Thi-Ngoc Nguyen[1,2] · **Yasushi Kiyoki**[1]

Received: 19 January 2017 / Accepted: 13 March 2017

Abstract Many multimedia retrieval tasks are faced with increasingly large-scale datasets and variously changing preferences of users in each query. There are at least three distinctive contextual aspects comprised to form a set of preferences of a user at each query time: content, intention, and response time. A content preference refers to the low-level or semantic representations of the data that a user is interested in. An intention preference refers to how the content should be regarded as relevant. And a response time preference refers to the ability to control a reasonable wait time. This paper features the dynamic adaptability of a multimedia search system to the contexts of its users and proposes a multicontext-adaptive indexing and search system for video data. The main contribution is the integration of context-based query creation functions with high-performance search algorithms into a unified search system. The indexing method modifies inverted list data structure in order to construct disk-resident databases for large-scale data and efficiently enables a dynamic pruning search mechanism on those indices. We implement a frame-wise video navigation system as an application of the indexing and search system using the a 2.14 TB movie dataset. Experimental studies on this system show the effectiveness of the proposed pruning search method when dealing with dynamic contexts and its comparative high search performance.

✉ Diep Thi-Ngoc Nguyen
chupi@sfc.keio.ac.jp

Yasushi Kiyoki
kiyoki@sfc.keio.ac.jp

[1] Graduate School of Media and Governance, Keio University, 5322 Endo, Fujisawa, Kanagawa, Japan

[2] Faculty of Information Technology, University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam

Keywords Large-scale multimedia retrieval · Context-dependent search · Frame-wise video navigation · Inverted index · Controllable response time

1 Introduction

While to end users, search is the most basic and quickest way to find information, at the core of a search engine, it is the indexing and search algorithms that work cleverly to meet that information need. Large-scale multimedia data challenge a search engine in several ways: organizing the contents of unstructured data; indexing large amount of records into databases; interpreting and adapting to changing contexts of users at query time; and quickly returning relevant information from the databases to users.

The *information need* of users is the topic that has been widely studied in information retrieval field [1,7,8]. While in principle text retrieval including document retrieval or Web page retrieval and multimedia retrieval share some principal concepts and methodology, in practice multimedia search seems to be more complex and exploratory [1]. Spink et al. [42] stated that "multimedia searching appears to require greater interactivity between users and the search engine." Comparing to the general Web page search, multimedia retrieval shows "a significant increase in the number of query terms, search session length, query reformulations, and number of search results clicks" [1].

Those differences can be explained by addressing the represented contents in a full-text document comparing to a multimedia datum. The contents of a document are often represented by the set of its words, whereas the contents of an image, for example, are often more unstructured and can vary depending on the low-level features chosen to represent it such as color, shape, or texture features. This gap between a

Published online: 20 March 2017

low-level features and high-level semantics is often known as a *semantic gap*. Besides the semantic gap, recent researches in multimedia retrieval have also realized another gap, the *intention gap* between search intent of a user and the query at query time [19,45]. However, instead of defining explicitly what can be an *intention*, the existing methods often suggest to use the user's feedbacks (i.e., the evaluation of a result is relevant or irrelevant) as an implicit interpretation of the search intent.

In this paper, three distinctive contextual aspects comprised to form a set of preferences of a user at query time are defined and treated: content, intention, and response time.

Content of interest This aspect refers to the low-level features of an image or video, which include color, shape, texture.

Intention of search This aspect refers to how the content will be regarded as relevant. There are three kinds of intention preferences: "dominant," "similar," and "exact." A "dominant" intention preference signifies that the strong characterized features should be considered as more important; therefore, a result will be counted as a match if it contains these features with high values. A "similar" intention preference indicates a desire to find similar data to the input datum given a preferred content. An "exact" intention preference suggests a search for only exactly matched data given the input datum and a preferred content.

Response time This aspect refers to the ability of users to control the response time of a search engine. Currently, most search systems run to completion and then return results to users. However, there are situations when the users want to choose a reasonable running time and want to get results by this time limit. This demand is supposedly raised when the dataset is large and the response time may be longer than an affordable wait time of a user.

Different persons, or the same person under different circumstances, may have different interest under one query. For a same input image in content-based image search, for example, one person may be more interested in the colors while another may be more interested in the shapes of objects in the input. Likewise, another person may wish to search for images that contain the vivid red colors as those are in the input image ignoring other light red colors or other colors. Only those returned images that match the user's interest and intention are evaluated as "relevant." This state of affairs challenges a search system to be able to treat those preferences and return reasonable results.

This paper features the dynamic adaptability of a multimedia search system and proposes a new indexing and search system for large-scale video data. The indexing method modifies inverted list data structure in order to construct disk-resident databases for large-scale data and efficiently enables a dynamic pruning search mechanism on those indices. The search method works adaptively to the preferences set up by users as input at query time. It reflects users' content preferences by selecting an appropriate subset of inverted lists and then adapts to the intention preferences by prioritizing indexes to search and initializing starting points to search for candidates. The searching for high-possibility relevant candidates is based on a greedy strategy to maximize the impact of importance content and the relevance of content measured by a similarity function.

The main contribution of this paper is an integrated context-based search system for multimedia with two important features: (1) a new disk-resident database construction to index low-level features of multimedia data using value-sorted inverted indices and (2) a dynamic pruning search method that adapts to multiple preferred contexts of users. We implement a frame-wise video navigation system using a 2.14 TB movie dataset and use it to assess the performance of our proposed search method comparing to other state-of-the-art algorithms such as KD-tree, ball tree, and local sensitive hashing (LSH forest).

The outline of paper is as follows: Sect. 2 describes related works and the originality of our methods comparing to them, Sect. 3 is the main section of this paper, in which our methodology is discussed after an introduction of the geometric intuition. In Sect. 4, we discuss the implementation of the video navigation system and the experimental studies on this system. Sections 5 and 6 discuss overall evaluations and conclude the paper.

2 Indexing and retrieval for large-scale datasets

2.1 Inverted indexing

Inverted index is a optimized data structure that facilitates efficient retrieval. This data structure is broadly used in information retrieval (IR) tasks where target data are documents. An inverted index includes a list of *inverted lists* for each *word* (or *term*). Each inverted list is a list of identifiers of the documents containing that *term*. The answers to a query "find the documents where word X occurs" can be retrieved quickly by looking up at the inverted list X. When querying one or more terms, documents are retrieved by looking up indexes of corresponding terms, and they are processed by computing their vectors of word frequencies and ranked to return as closer distance to the query. In either case, the time and processing resources to perform the query are dramatically improved.

Using inverted index structure for other multimedia information retrieval tasks such as for image and video is currently

Table 1 Categories of methods use inverted index

Viewpoint	Categories
Organization	In-memory, **on-disk**
List structure	Document sorted, frequency sorted, dual sorted, **value sorted**
Content representation	**Low-level features**, "visual words"
Search method	Pruning from head, **context-adaptive pruning**
Application	Text retrieval, image retrieval, **video retrieval**

The bold methods are used or newly proposed in this research

investigated, e.g., [13,20,34,41,43]. Researches in this direction treat either low-level features of images such as color and texture in [43] or "visual words" of quantized descriptors such as [41] as set of terms to be indexed. By searching time, the algorithms get out all the images which contain features appear in query and adding them to the pool of candidate images to be ranked.

There are two common ways to organize the inverted list: *document sorted* [20,41] and *frequency sorted* as stated in [39]. Document-sorted inverted list means the list of documents are sorted by document identifiers, commonly by an increasing order. Frequency-sorted list means the list of documents are sorted by the frequency of the term occurring in the documents, commonly by a decreasing order. The document-sorted technique is useful for Boolean queries, whereas the frequency-sorted technique is useful for ranked document retrieval. A combined technique named *dual sorted* [36] can be used to maintain a single data structure that offers both two orderings of frequency-sorted and document-sorted inverted lists.

From storage viewpoint, the inverted index is stored either in-memory [41] or on-disk [20,41] and either be compressed [26,44] or not. For large-scale data, it's more appropriate to use on-disk storage so that the memory consumption can be reduced significantly by transferring only needed part of each inverted list from disk [2].

Table 1 summarizes categories of methods that use inverted index: organization of inverted index (either in-memory or on-disk), structure of inverted index (either document sorted or frequency sorted), representation of semantic content (either using low-level features or visual words), and its applications (either text, image, or video retrieval). We present in this paper an on-disk, modified frequency-sorted (called "value sorted") inverted indexing method, which is as an improved method to support video retrieval by dynamic path finding search method with a guaranteed performance.

2.2 The curse of high dimensionality

Searching in a large amount of multimedia data whose semantic features are represented in high-dimensional metadata has been main objective of many researches. There is a phenomenon called "curse of dimensionality" that arises when analyzing data in high-dimensional spaces. This phenomenon suggests that close objects might get separated by a partition boundary when partitioning the space. The straightforward and common solution to tackle this problem is to reduce the number of dimensions (or features) that are used to represent the multimedia data such as feature selection methods [9] or dimension reduction methods [21]. However, reducing the number of dimensions causes reduced "subtleness" expressed in the original data, which is, most of the time, what users are looking for.

It is observed that despite the high dimensionality of data, the number of dimensions with respect to a content preference can be relatively small. For example, even if we use thousands of colors to represent the content of image data, it is more likely that we want to search for some particular colors at query time. For that reason, this paper suggests to manipulate the dimensionality of data in a context-adaptive way.

2.3 Fast nearest neighbors retrieval

Many researches have utilized approximate nearest neighbors problems to solve large-scale multimedia retrieval tasks. Content-based image retrieval systems have been able to manage collections having sizes that could be very difficult years back. Most systems can handle several million to hundred million images [15,20,23,27,35,46], billions of descriptors [22,28,35], or address web-scale problems [3,16,29,46], and so on.

The approaches to overcome large-scale datasets and high-dimensional presentation of data vary from many ranges, but in general can be categorized as follows:

- Cluster based and its derivatives, e.g., [18,29,35,40]
- Locality-sensitive hashing, e.g., [29,31]
- Tree-based, e.g., [27,28,30,37,38]
- MapReduce paradigm, e.g., [35]

Most of the approximate high-dimensional nearest neighbor search methods based on the above categories are based on some kinds of segmentation of the data collection into groups named *clusters* or partition the data space into areas so that *close* data are indexed in the same *cluster*, or with the same *hash code*, or at the same *node leaf*.

It is easy to see that the space partitioning is done at index time in these methods, but using space partitioning at search time is our original proposal. The importance of treating the

data space dynamically depending on the contexts at query time has been recognized early [25,34]. However, the current situation of emergence of large-scale data and demanding attention to users' preferences when searching for information have made this idea more alerted. Additionally, the computational resources as well as the storage resources that we have nowadays enable us to extend this research direction further.

3 Multicontext-adaptive search method

3.1 Geometric intuition

Suppose we have a dataset distributed in a high-dimensional space and when a contextual preference is given, a subspace of a less number of dimensions is selected as the new space for calculation, which is either searching or ranking or more complex integrated tasks. The intuition is shown in Fig. 1a. This idea emphasizes the importance of dynamic recognition of contexts for computing and has been early recognized and proposed by Kiyoki et al. [24]. The context-based subspace selection has an intrinsic ability to alleviate the curse of high dimensionality as discussed in Sect. 2.2. Moreover, it also yields some degrees of interpretability of semantic computing processes.

Assume that we are interested in two *attributes* of data and want to proceed a search for information with respect to an existing *preference*. In this case, the data and query from high-dimensional space are projected into a two-dimensional space as shown in Fig. 1b. We will only treat numerical values of attributes of data so that the subspace is a subspace of real numbers in which orders between every two values can be made.

In Fig. 1b, the *content preference* is supposedly expressed by the position of the *query* point. At this moment, we can have at least three intentions to *order* other data points and have them listed as the results of a search problem. The first intention is to find the answers to the question: "which is the closest point to the query?" The second intention is to find the answers to the question: "which points are close to the query?" And the third intention is to find the answers to the question: "which points have a large sum of values in some direction?" The first and second questions are common in information retrieval, while the third question expects more exploratory answers.

Intuitively, if a data point is close to the query, it is close to the query in any direction. Based on this observation, we can approximate the locations of *candidates* as some areas that are remote from the query points by some small distances as shown in Fig. 1c. The original idea has been proposed by Kiyoki et al. [25] who applied this intuition for document retrieval problems. Figure 1c shows how we can quickly approximate some candidates , "c" and "a" by looking at the areas that are less remote from the query point in both directions. Although it can be seen that the point "a" is unlikely to *closer* to the query point comparing to other points like "e" and "b," this kind of *errors* is hopefully resolved in some next searches starting from the location of "c" on the horizontal and from the location of "a" on the vertical axis.

Figure 1d shows that the search for *close* candidates depends on the query's location. Firstly, it assumes that the larger value in one direction of a datum implies the greater importance. Therefore, in this case of the new query, the vertical direction is prioritized higher when looking for the candidates. Secondly, this idea suggests a notable search technique that can adapt to the context to switch its own search path for results in a meaningful way.

The next question is where can we find the closest point to the query without any error. The intuitive location to find such point is the covering rectangle as shown in Fig. 1e. This rectangle covers the least distance intervals from the query point in all directions. If we continue searching for candidates like in Fig. 1c, there will be a moment we find at least one candidate appears in both directions and we have a rectangle that covers already found candidates. The closest point will be among these, not any outside the rectangle.

The third question is "which points have a large sum of values in some direction." The answers are expected by finding candidates starting from the points which have very large values in one direction as shown in Fig. 1f. This intuition has been originally applied for image data in the work of Miyagawa et al. [34]. In Fig. 1f, we see that we re-rank the data points on each direction based on a descending order of values. On the first prioritized feature direction, we find point "*g*" which has the highest value and on the second feature direction, we find the point "*d*." They are the starting points to continue looking for the candidates. It is important to note that this intuition works as a *greedy* strategy and does not guarantee that the final order based on the sums of values in both directions. The sums of values in both directions will be done giving us the final order of candidate points. It is to be shown by experiments in Sect. 4.5 that this intuition can fail badly when the number of preferred features is increased.

3.2 Generalized indexing and search method

We propose a multicontext-adaptive search method based on the geometric intuition discussed in Algorithm 1. The logic of the search method contains five steps: (1) reflecting content preference onto query vector, (2) prioritizing directions for searching, (3) initializing starting points of search, (4) iteratively finding matching candidates, and (5) ranking candidates and returning top results to users.

In this algorithm, the content preference is expressed as a vector p and reflected during computation in lines 1 and 15.

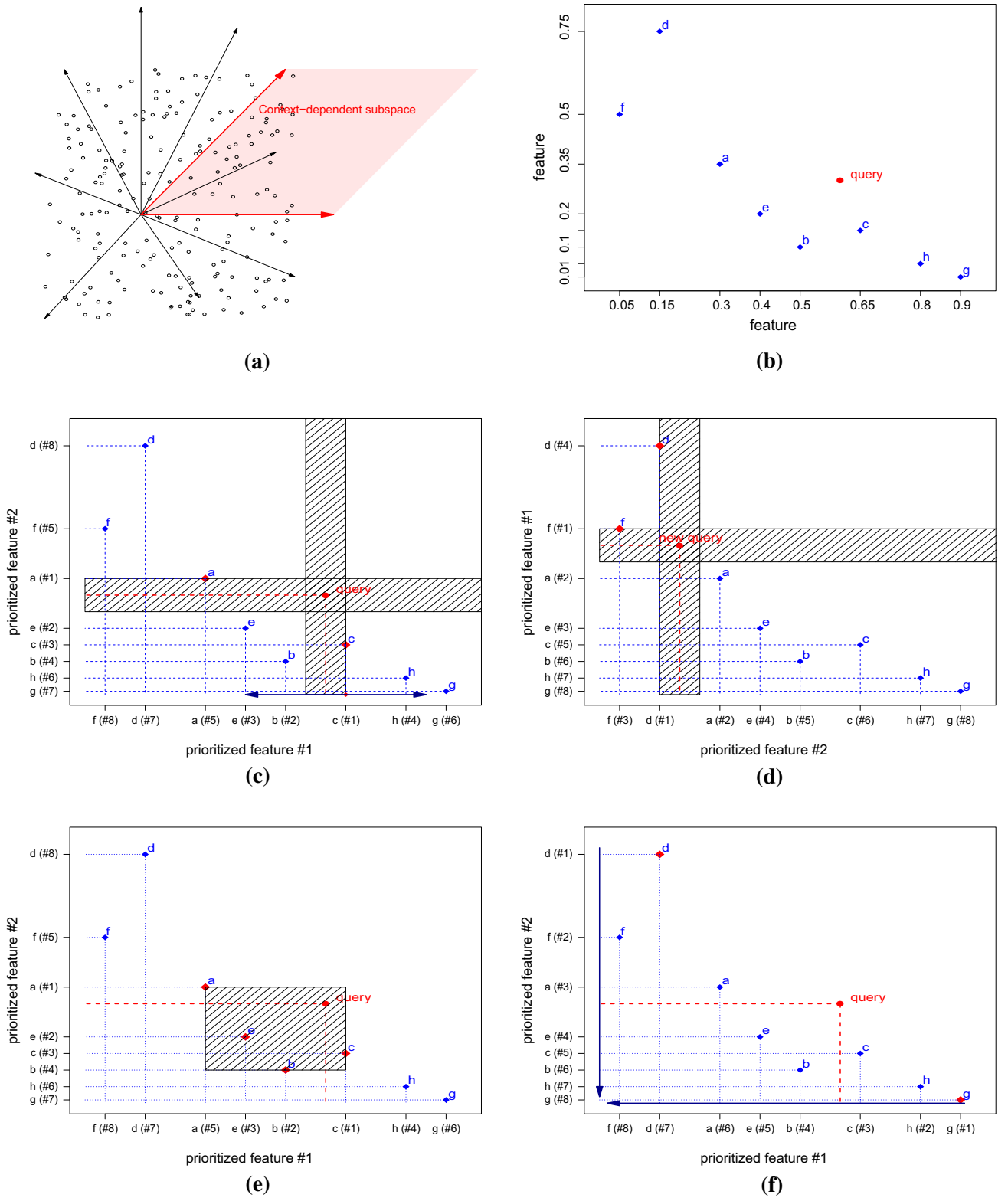

Fig. 1 Geometric intuition of the multicontext-adaptive search strategy. **a** Data on high-dimensional space and subspace selection. **b** Sample data and query on the selected subspace $\mathbb{R}^2$. **c** Approximating locations of the first *similar* candidates. **d** Location approximation depending on the query's location. **e** Location of *exact-match* candidates. **f** Locations of *dominant* candidates

Algorithm 1 Multicontext-adaptive pruning search mechanism

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise;
- A preference of intention $\mathbb{I} \in$ *dominant*, *similar*, *exact*;
- Optional: Maximum number of priorities m, by default $m = 5$;
- Optional: Timelimit t, by default $t = 1(second)$ or *None* for *exact* matching;
- Optional: Number of results R, by default, $R = 20$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

1: [*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
2: [*Step 2: Prioritize features*] A permutation function σ of the set $\{1, \ldots, d\}$ that sorts elements of vector q in a descending order: $\left(q_{\sigma(1)} \geq \ldots \geq q_{\sigma(d)}\right)$.
3: [*Step 3: Initialize starting points*] Accumulator vector $v \in \mathbb{R}^m$:
4: **if** $\mathbb{I} =$ *dominant* **then**
5: $\quad v \leftarrow \left(\max_{x \in \mathbf{X}} x_{\sigma(1)}, \ldots, \max_{x \in \mathbf{X}} x_{\sigma(m)}\right)$.
6: **else**
7: $\quad v \leftarrow \left(q_{\sigma(1)}, \ldots, q_{\sigma(m)}\right)$.
8: **end if**
9: [*Step 4: Find candidates*]
10: List of candidates $\mathbf{C} \leftarrow \{\}$
11: **while** time limit t is not reached **do**
12: $\quad$ **for** i **from** 1 **to** m **do**
13: $\quad\quad c \leftarrow \arg\min_x \left|x_{\sigma(i)} - v_i\right|$ for all $x \in \mathbf{X}$ not yet in $\mathbf{C}$.
14: $\quad\quad v_i \leftarrow c_{\sigma(i)}$
15: $\quad\quad score = relevant(q, c \odot p)$. ▷ see section 4.1.4.
16: $\quad\quad$ **if** $\mathbb{I} =$ *exact* and $score = 0.0$ **then**
17: $\quad\quad\quad$ **return** c
18: $\quad\quad$ **end if**
19: $\quad\quad$ Update $\mathbf{C}$ with candidate c and $score$.
20: $\quad\quad$ Sort $\mathbf{C}$ by an appropriate order of scores. ▷ see section 4.1.4
21: $\quad$ **end for**
22: **end while**
23: [*Step 5: Return top results*]
24: **return** top R of $\mathbf{C}$ to users.

The intention preference is processed at conditional branching in lines 4 and 5. The response time limit preference is controlled in the *while* loop from line 11.

The iterative finding candidates in each direction of prioritized features are repeated as in line 13. In order to speed up this process, we propose to index each feature as an inverted list with value sorted. Moreover, all indices and reference data will be written to binary files on disk by data blocks so that each can be accessed by random access when needed.

Using a value-sorted inverted list, the first candidate in each direction can be found quickly using binary search and the next candidates can be found incrementally from the index of the first candidate.

Algorithm 1 describes the generalized pruning search mechanism which is the main proposal of this paper. In Sect. 4, we implement three concrete algorithms based on this generalized mechanism which are called "combinatorial search algorithm," "exact-match search algorithm," and "Maxfirst search algorithm." The geometric intuitions for these algorithms are shown in Fig. 1c, e, f, respectively.

Table 2 shows the classification of contexts in which each proposed search algorithm can be applied. The content preferences are introduced following the feature types of video data which are used in the experimental video navigation application system in Sect. 4. They include "color," "shape," "texture," or combination of those features. However, these content preferences are not limited to these types. Any feature type that can be ordered is applicable to the search algorithms.

It is also to note that the "interruptible" preference regarding the response time limits introduced in Table 2 refers to an interactive behavior of a user to abort the search process at anytime it is running. This interaction between users and search systems can provide an interesting usability in which the users of search systems can receive "better" results while the systems are running and decide at their will when to stop. Although this is not shown in Algorithm 1, it is possibly implemented by modifying the main *while* loop (line 11) of each corresponding algorithm.

4 Experiments

In this section, we introduce a frame-wise video navigation system and described some experimental studies based on this system.

4.1 Frame-wise video navigation system architecture

The overall system architecture is shown in Fig. 2. It includes six modules which are (1) data collection, (2) frame extraction, (3) feature extraction, (4) indexing, (5) search, and (6) ranking and display. The data collection module collects image and video data from many resources which can be either from personal collections of photos and videos or from the Internet such as YouTube (video data) or Flickr (image data) and many others. The frame extraction module works only with video data, and its purpose is to reduce the number of frames of a video to be indexed. This module contains three main steps: for each video record, it samples the video frames from the beginning by a fixed time (by default, $\delta t = 1$ s) and applies a threshold-based scene detection algorithms to detect scenes in the video. Finally, the thumbnails of the detected scene frames are extracted from the video and temporally saved as key frames of the video to be indexed.

The scene detection algorithm detects a new scene comparing to previous scene and keeps only the starting frame in the consecutive frames of a scene as a representative frame for the scene. The frame at 0 s (the beginning of the video) is the first presentative frame of a video by default. A frame is said to be on new scene if it is less similar to the current

Table 2 Classification of applicable contexts of three proposed search algorithms

Contextual preference		Maxfirst search	Combinatorial search	Exact-match search
Content	Color	Yes	Yes	Yes
	Shape	Yes	Yes	Yes
	Texture	Yes	Yes	Yes
	Combination	Yes	Yes	Yes
Intention	"Dominant"	Yes	No	No
	"Similar"	No	Yes	Yes
	"Exact-match"	No	No	Yes
Response time	Limited	Yes	Yes	No
	Unlimited	Yes	Yes	Yes
	Interruptible	Yes	Yes	No

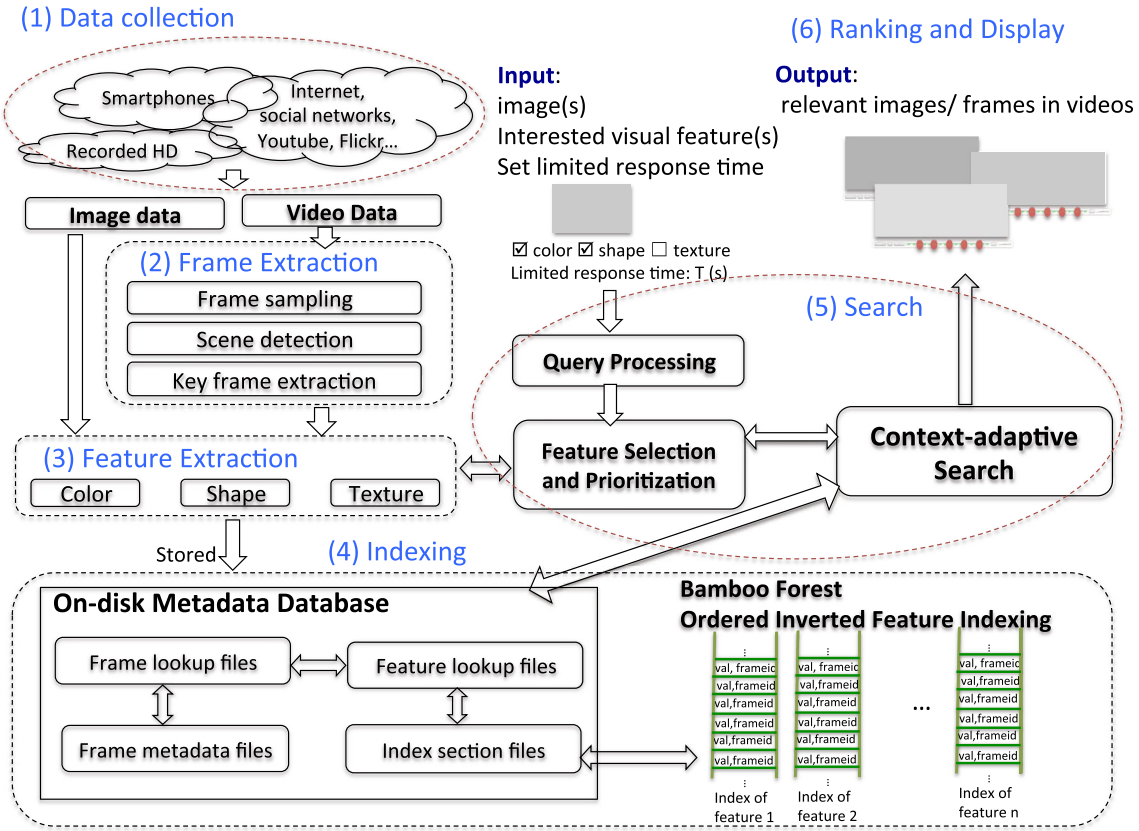

Fig. 2 Overall architecture of the frame-wise video navigation system with multicontext-adaptive indexing and search

frame by a fixed threshold. We extract the CEDD [11] integrated feature vector for each frame and use cosine distance to calculate the similarity between two frames. The cosine similarity is defined by

$$cosine\_similarity(x, y) = 1 - \frac{x \cdot y}{\|x\| \, \|y\|}. \quad (1)$$

where $\|x\|$ is the magnitude of a vector represented by frame x and $x \cdot y$ operation is the dot product between two vectors x and y.

The computed distance ranges from 1.0 to 0.0 in which 0.0 means two frames are *completely similar*, and 1.0 means two frames are *completely different*.

The feature extraction module implements conventional feature extraction methods to extract metadata of images and key frame thumbnails. The extracted features are color [17], shape [6], texture [32] and integrated features including FCTH [12], CEDD [11], and JCD [10]. The number of dimensions for color, shape, texture, FCTH, CEDD, JCD features is 63, 40, 60, 192, 144 and 168, respectively.

Table 3 Details of movie dataset used for experiments

	Movie dataset
Number of videos	806
Total length	597 h
Total storage size	2.14 TB
Total number of frames	2,150,428
Total number of scene frames	485,337
Reduced rate	22.60%

Table 4 Six databases constructed from the movie dataset and their storage size

Database	Size of database
10k	105M
50k	517M
100k	1.0G
200k	2.0G
300k	3.0G
500k	4.9G

4.1.1 Dataset

We use a 2.14 TB movie data as the dataset for the system with details shown in Table 3. The dataset is a personal collection of movie DVD including various genres but mainly of animation, drama, and documentary. The total view length is 597 h for 806 movies meaning the averaged length of a movie is 45 min. We run the scene detection algorithm discussed earlier to reduce the number of frames from 2 millions to about 500 thousands scene frames.

4.1.2 Video indexing

We divide the scene frames into six subsets, which include 10, 50, 100, 200, 300, and 485 thousand frames, and construct the respective database of inverted lists for each. The six databases are named "10k," "50k," "100k," "200k," "300k," and "500k," respectively. Table 4 shows the six databases and their on-disk storage size. Although the size of a database is large, at query time, only a part of the database will be used. Moreover, we assume that the concern of storage size is trivial for the task at hand, which is to perform adaptive search based on context preferences.

4.1.3 Search algorithms implementation

We define three concrete algorithms based on the general search algorithm 1. The first algorithm is named "Maxfirst" algorithm, which corresponds to the context where the *intention* preference is "dominant." The second search algorithm is named "combinatorial" algorithm, which corresponds to the context where the *intention* preference is "similar." The third search algorithm is named "exact-match" algorithm, which corresponds to the context where the *intention* preference is "exact." Section 3.2, and Table 2 describes the detailed descriptions of the three proposed search algorithms.

We implement our search method using Python programming language version 2.7.12[1] on two computers: a desktop and a laptop. The desktop computer, which is a laboratory server, has a faster processor and larger memory, but databases are stored on its hard disk drive (HDD). Its configurations are: CentOS Linux release 7.1.1503 (Core), 32 CPU Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz. The laptop computer is a general-purpose computer, which is a MacBook Air 1.7 GHz Intel Core i7, 8 GB 1600 MHz DDR3 with OXS Yosemite operating system. On this laptop, the databases are stored on an external SSD disk (Samsung 850 EVO 500 GB), connecting to the laptop via a USB 3.0 cable.

4.1.4 Baseline setting

We use top 20 results returned by a brute force algorithm as the baseline for a query search. When searching for "similar" or "exact," the relevant similarity will be calculated by the cosine similarity. In this case, the candidates will be ranked by an ascending order of scores. But when search for "dominant," the sum of corresponding prioritized features is calculated as relevant scores and the ranks of candidates will be on a descending order of those scores.

For each database, we randomly select 50 frames as input images and then run the brute force algorithm for each of six feature kinds. In total, we have 300 queries for each database.

4.1.5 R-Precision criterion

We use "R-precision" as the measurement of precision of an algorithm comparing to the baseline. The definition of R-precision is as follows [14]: For a given query topic Q, R-precision is the precision at R, where R is the number of relevant data for Q. In other words, if there are r relevant frames among the top-R retrieved frames, then R-precision is $\frac{r}{R}$. Chistopher et. al [33] showed a high correlation of this measure to the well-known mean average precision(MAP). In our experiments, we set $R = 20$.

4.2 Running time and confidence of finding an exact match

In this experiment, we examine the running time of our exact-match algorithm at different settings of database size and number of prioritized features used ($\#features = \{2, 5, 10, all\}$). Since the result of exact-match is only one

[1] https://www.python.org/.

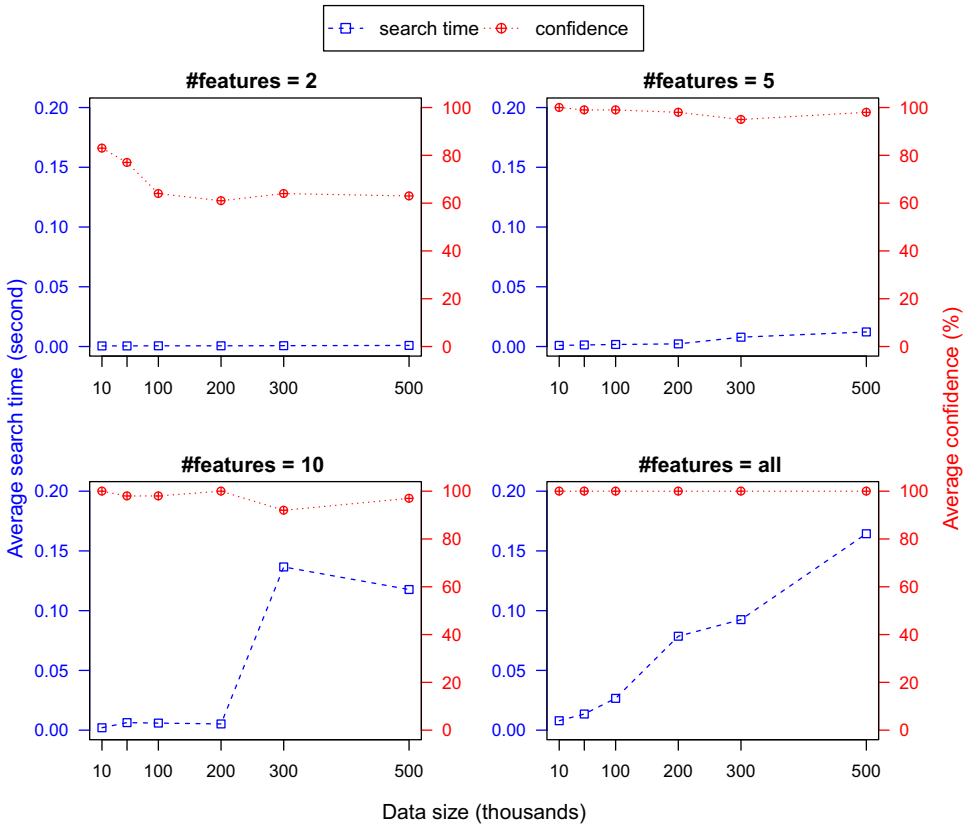

Fig. 3 Performance of the proposed exact-match search algorithm by different data sizes, and settings of number of prioritized features (#*features*)

result ($R = 1$), instead of using R-precision, we define "average confidence" as a measure of the percentage when the algorithm returns a right exact match.

The results are shown in Fig. 3. In this figure, we can see the correlation between the average search time and the data size and the correlation between the number of prioritized features and the average time and average confidence. It is not surprising that when all features are used, the exact-match search algorithm returns a result with 100% confidence that it is the exact match. Although the search time is high and increases proportionally to the data size, it is reasonably low (about 0.15 s).

On the other hand, we can reduce the average search time by using a smaller number of prioritized features (let m denote this number) although if applying this, the returned result can only be guaranteed to be the right match with some probability. When $m = 2$, which is the lower bound of m in the exact-match search algorithm, we see the average search time is very low and stable regardless of data size. However, the average confidence is only about 60%.

As m increases, the average search time increases but also the average confidence. When $m = 5$ and $m = 10$, we see that the average confidences are almost the same (slightly higher in the case $m = 5$) but the average search time is quite different, especially when the size of data is large (greater than 300,000).

4.3 Comparative search time and precision versus data size

In this second experiment, we compare the average search time for one query and the average R-precision of the proposed combinatorial search algorithm with the brute force algorithm, and two spatial tree algorithms (KD-tree [5] and ball tree [37]) and a hashing algorithm (local sensitive hashing (LSH) forest [4]) for different sizes of data.[2] The result is shown in Figs. 4 and 5.

In Fig. 4, the KD-tree and LSH forest algorithms response remarkably quick (less than 0.5 s for all databases); however, their precisions as in Fig. 5 are also noticeably poor: less than 50% (LSH forest) or about 70% (KD-tree). On contrary, ball tree algorithm obtains very high precision (Fig. 5) but responses comparatively as the brute force algorithm (Fig. 4).

In Figs. 4 and 5, we see the proposed combinatorial search algorithm responses with reasonably high precision and short average search time. Although the average search time proportionally increases as the size of data increases, comparing to ball tree and brute force algorithms, this search time is

[2] (1) KD-tree, ball tree, and LSH forest algorithms are implemented using *scikit learn nearest neighbors* modules. Source: http://scikit-learn.org/stable/modules/neighbors.html. (2) For large datasets, the maximum recursion limit of a tree algorithm can be exceeded; therefore, set it before running to 10,000.

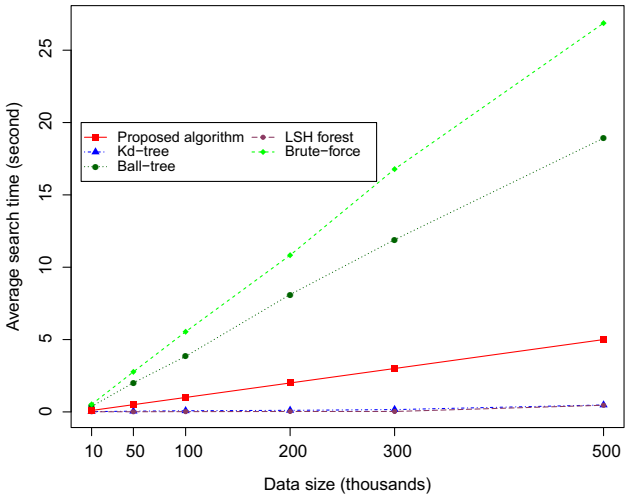

Fig. 4 Comparative average search time of the proposed combinatorial search algorithm to others search algorithms with increasing size of data

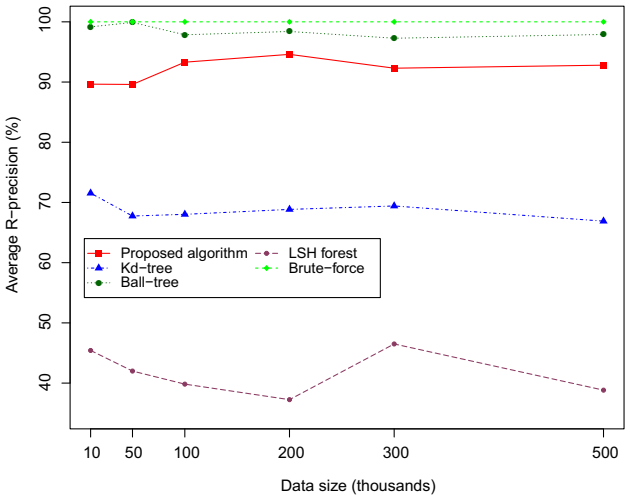

Fig. 5 Comparative average R-precision of the proposed combinatorial search algorithm to different search algorithms with increasing size of data

fairly acceptable. It is to note that the response time of the combinatorial search algorithm can be controlled by setting a time limit for it. In Fig. 4, the average search time is set so that the corresponding precision in Fig. 5 is above 90%.

4.4 Precision of combinatorial search algorithm

The third experiment examines the performance of our proposed combinatorial search algorithm running with different settings of number of prioritized features (m) and response time limits and database size. The results are shown in Fig. 6. In this figure, the overall precision drops as the size of database increases but gradually obtains reasonable precision as the limit time increases.

By comparing the average precision by different settings of m, we can see that a larger m does not return better precisions. It is because when m is large, the algorithm has to look for the candidates from more number of feature indexes, which can be useful until a some value of m. As shown in Fig. 6, when increasing m from 5 to 10, we obtain some better precisions for large databases ("300k" and "500k") but not when m is larger than 15.

Apparently, the performance of the combinatorial search algorithm is affected by the IO access speed as shown in Fig. 7. In this figure, the queries are set with $m = 10$ and done on two computers. Figure 7 shows that the search algorithm performs better with databases on storage devices that provides fast IO accesses.

4.5 Precision of Maxfirst search algorithm

In this experiment, the performance of the proposed Maxfirst search algorithm will be examined by setting its parameters with various values of response time limits, numbers of prioritized features, and sizes of Bamboo forest database. The response time limits are 0.1, 0.5, 1.0, 2.0, 3. 0, and 5.0 s. The number of prioritized features (m) is one from the set ($\#features = \{2, 5, 10, 15\}$). And the databases are described in Table 4.

The result of this experiment is shown in Fig. 8. In this figure, the performance is evaluated based on the R-precision. In this figure, the proposed Maxfirst search algorithm performs more stable and efficient when the number of prioritized features (m) is small or the limited response time is large. As m increases, the average precisions decrease and apparently correlate with the size of data and the limited response time. This can be explained by the sparse distribution of the values of features when the number of features is large, meaning the "dominant" characteristics are lost and shared by many dimensions.

When m is small, increasing time limit does not necessarily increase the precision unless the time limit is set to infinite and in this case the Maxfirst search algorithm works equivalently as a brute force algorithm. However, when m is large, increasing time limit can increase the precision to some limits.

5 Discussion

The previous sections have discussed the performance of the proposed search algorithms by several settings of their parameters. It is not surprising that in all three search algorithms, increasing the time limits gains better precisions. However, the choice of the number of prioritized features (m) is critical in order to avoid redundant computations while aiming for higher performance.

In the case of exact-match finding (using the exact-match search algorithm), if the goal is to find the guaranteed right

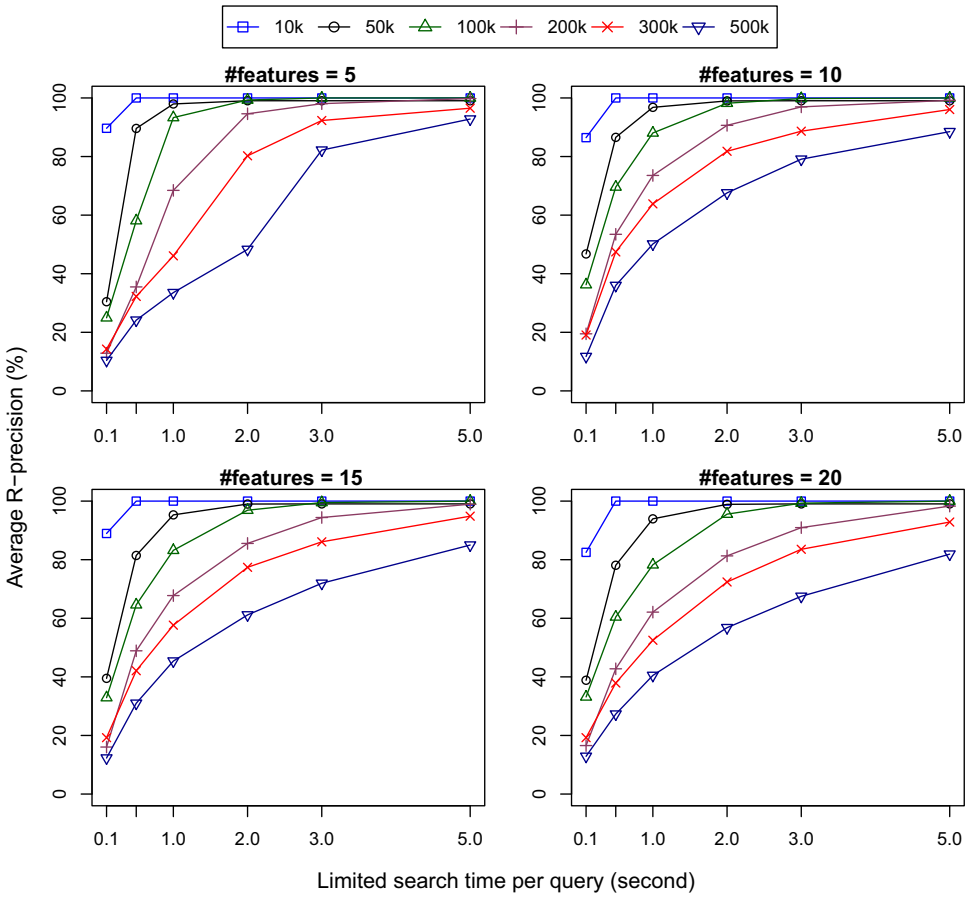

Fig. 6 Precision of the proposed combinatorial search algorithm by different response time limits, number of prioritized features, and database size

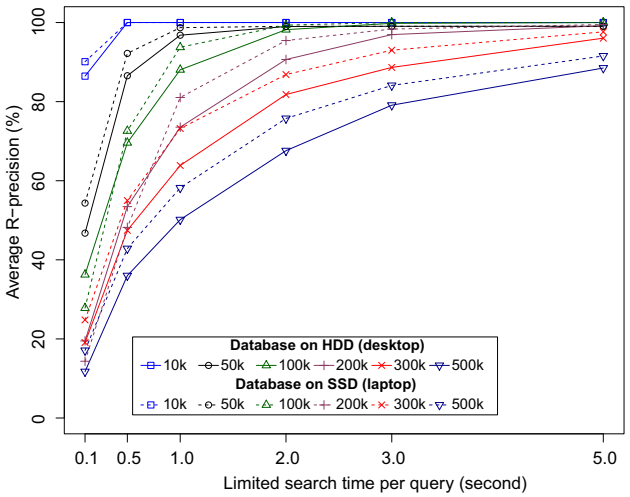

Fig. 7 Performance of the proposed combinatorial search algorithm implemented on two computers of different computing and IO access speeds

answer (100% *confidence*), then m should not be chosen less than the number of features that the content preference indicates. In other words, all dimensions in the selected subspace will be used. However, if the goal is to find the right answer with some (high) probability and in a short time, then m can be set for a value not very low or high. Figure 3 shows $m = 5$ is a reasonable setting.

In the same way for similar matches finding (k-NN problem) using the combinatorial search algorithm, a low or high value set for m does not gain the higher performance. Figure 6 shows $m = 10$ is a reasonable setting.

On the contrary, the Maxfirst search algorithm performs stably when m is low and when m is high, it can only gain higher precision with higher settings for time limits.

The next comment regards to the indexing time and size of indexed databases. In Sect. 4.3, only the average search time per query of each algorithm was used to compare the performance of the proposed combinatorial search with other search algorithms. It is to note that the brute force algorithm performs similarity calculation directly on the data without any pre-indexed database. KD-tree, ball tree, and LSH forest search algorithm index the data into their defined data structures (in-memory database) and query for results on those database. The indexing time for each algorithm for each size of data was not counted in the search time. Likewise, the proposed search algorithm works with the pre-indexed on-disk databases and the index time was not taken into account. The sizes of the databases have also not yet been concerned.

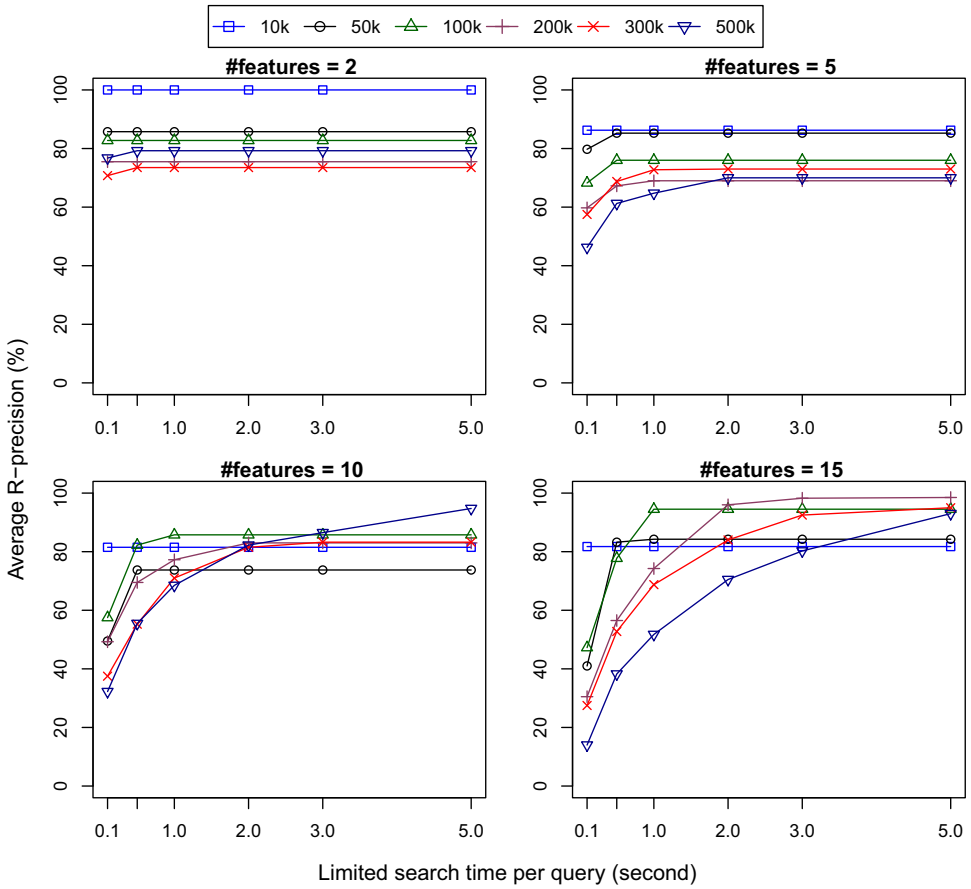

Fig. 8 Average precision of the proposed Maxfirst search algorithm by different settings of the number of prioritized features, the size of database, and the limited response time

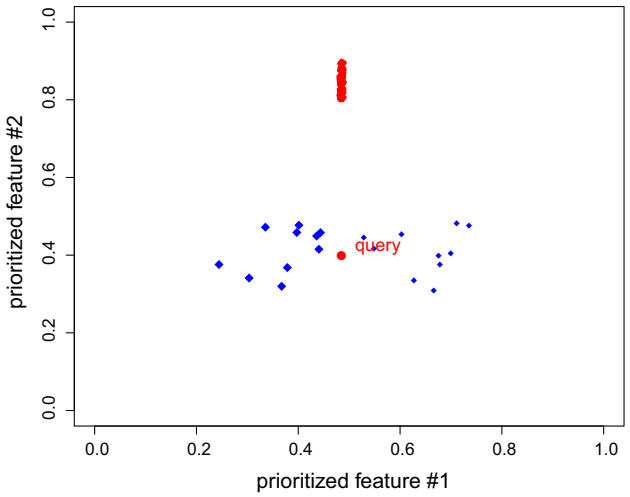

Fig. 9 A typical bad case of the proposed heuristic search strategy in which the searching process wastes time at "local" candidates

Finally, we discuss "when do the proposed algorithm suffer." A typical bad case for the multicontext-adaptive search algorithms is intuitively shown in Fig. 9. The query point has x-axis value larger than y-axis value; therefore, the x-axis is prioritized higher than the y-axis. A search process based on a heuristic search strategy described in Sect. 3.1 will start checking the candidates which has x-axis value close to the x-axis value of the query point. In this case, those candidates are the red points. Intuitively, the actually distances from the query to those data points are larger than those to some blue data points. In other words, the search process wastes time at some "local" candidates.

This kind of bad cases is supposed to happen in several special situations: (1) the size of dataset increases, (2) there are too many duplicated data in the dataset, or (3) there are non-discernible features used for representing data.

The degree of a proposed search algorithm suffering from this "local candidate" phenomenon depends on the distribution of data on the search space. However, it is likely that the Maxfirst search algorithm will suffer more than other two algorithms.

6 Conclusion

In this paper, we proposed a new indexing and pruning search system for large-scale video data that provide users more controls of their preferences at query time: content, intention, and response time. The proposed system is an integration of multicontext-adaptive querying methods and high-

performance search algorithms for multimedia retrieval. We implemented a frame-wise video navigation system and used it to study the performance of our methods. The experiments show the effectiveness of the heuristic search strategies on modified inverted list databases that can obtain comparatively high precisions in a relative low response time comparing to other state-of-the-art methods including tree-based and hashing methods. The experiments also show the scalability of the system for large-scale datasets.

Acknowledgements The authors would like to acknowledge the continuing support of the H28 Keio University Doctorate Student Grant-in-Aid Program and the Taikichiro Mori Memorial Research Fund 2016.

References

1. André P, Cutrell E, Tan DS, Smith G (2009) Designing novel image search interfaces by understanding unique characteristics and usage. In: IFIP conference on human–computer interaction. Springer, pp 340–353
2. Anh VN, Moffat A (2006) Pruned query evaluation using pre-computed impacts. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp 372–379
3. Batko M, Falchi F, Lucchese C, Novak D, Perego R, Rabitti F, Sedmidubsky J, Zezula P (2010) Building a web-scale image similarity search system. Multimed Tools Appl 47(3):599–629
4. Bawa M, Condie T, Ganesan P (2005) Lsh forest: self-tuning indexes for similarity search. In: Proceedings of the 14th international conference on World Wide Web. ACM, pp 651–660
5. Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
6. Bosch A, Zisserman A, Munoz X (2007) Representing shape with a spatial pyramid kernel. In: Proceedings of the 6th ACM international conference on image and video retrieval. ACM, pp 401–408
7. Broder A (2002) A taxonomy of web search. SIGIR Forum 36(2):3–10. doi:10.1145/792550.792552
8. Case DO (2012) Looking for information: a survey of research on information seeking, needs and behavior. Emerald Group Publishing, Bradford
9. Chandrashekar G, Sahin F (2014) A survey on feature selection methods. Comput Electr Eng 40(1):16–28
10. Chatzichristofis S, Lux M, Boutalis Y (2009) Selection of the proper compact composite descriptor for improving content based image retrieval. In: Proceedings of the 6th IASTED international conference, vol 134643. p 064
11. Chatzichristofis SA, Boutalis YS (2008a) Cedd: color and edge directivity descriptor: a compact descriptor for image indexing and retrieval. In: Computer vision systems. Springer, pp 312–322
12. Chatzichristofis SA, Boutalis YS (2008b) Fcth: Fuzzy color and texture histogram-a low level feature for accurate image retrieval. In: Ninth international workshop on image analysis for multimedia interactive services, 2008. WIAMIS'08. IEEE, pp 191–196
13. Chen DM, Tsai SS, Chandrasekhar V, Takacs G, Vedantham R, Grzeszczuk R, Girod B (2010) Inverted index compression for scalable image matching. In: DCC. p 525
14. Craswell N (2009) Encyclopedia of database systems, R-precision. Springer, Boston
15. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: IEEE conference on computer vision and pattern recognition, 2009. CVPR. IEEE, pp 248–255
16. Douze M, Jégou H, Sandhawalia H, Amsaleg L, Schmid C (2009) Evaluation of gist descriptors for web-scale image search. In: Proceedings of the ACM international conference on image and video retrieval. ACM, p 19
17. Graf F (2012) Jfeaturelib—a free java library containing feature descriptors and detectors. https://jfeaturelib.googlecode.com
18. Gudmundsson G, Jónsson B, Amsaleg L (2010) A large-scale performance study of cluster-based high-dimensional indexing. In: Proceedings of the international workshop on Very-large-scale multimedia corpus, mining and retrieval. ACM, pp 31–36
19. Hanjalic A, Kofler C, Larson M (2012) Intent and its discontents: the user at the wheel of the online video search engine. In: Proceedings of the 20th ACM international conference on multimedia. ACM, pp 1239–1248
20. Hare JS, Samangooei S, Dupplaw DP, Lewis PH (2012) Imageterrier: an extensible platform for scalable high-performance image retrieval. In: Proceedings of the 2nd ACM international conference on multimedia retrieval. ACM, p 40
21. Jacques J, Preda C (2014) Functional data clustering: a survey. Adv Data Anal Classif 8(3):231–255
22. Jégou H, Tavenard R, Douze M, Amsaleg L (2011) Searching in one billion vectors: re-rank with source coding. In: IEEE international conference on acoustics, speech and signal processing (ICASSP), 2011. IEEE, pp 861–864
23. Jégou H, Perronnin F, Douze M, Schmid C et al (2012) Aggregating local image descriptors into compact codes. IEEE Trans Pattern Anal Mach Intell 34(9):1704–1716
24. Kiyoki Y, Kitagawa T, Hayama T (1994) A metadatabase system for semantic image search by a mathematical model of meaning. ACM Sigmod Rec 23(4):34–41
25. Kiyoki Y, Kitagawa T, Miyahara T (1996) A fast algorithm of semantic associative search for databases and knowledge bases. Information modelling and knowledge bases VI. IOS Press, Amsterdam, pp 44–58
26. Konow R, Navarro G, Clarke CL, López-Ortíz A (2013) Faster and smaller inverted indices with treaps. In: Proceedings of the 36th international ACM SIGIR conference on research and development in information retrieval. ACM, pp 193–202
27. Lejsek H, Ásmundsson F, Jónsson BT, Amsaleg L (2009) Nv-tree: an efficient disk-based index for approximate search in very large high-dimensional collections. IEEE Trans Pattern Anal Mach Intell 31(5):869–883
28. Lejsek H, Jónsson B, Amsaleg L (2011) Nv-tree: nearest neighbors at the billion scale. In: Proceedings of the 1st ACM international conference on multimedia retrieval. ACM, p 54
29. Li X, Chen L, Zhang L, Lin F, Ma WY (2006) Image annotation by large-scale content-based image retrieval. In: Proceedings of the 14th annual ACM international conference on multimedia. ACM, pp 607–610
30. Liu T, Rosenberg C, Rowley HA (2007) Clustering billions of images with large scale nearest neighbor search. In: IEEE workshop on applications of computer vision, 2007. WACV'07. IEEE, pp 28–28
31. Lv Q, Josephson W, Wang Z, Charikar M, Li K (2007) Multi-probe lsh: efficient indexing for high-dimensional similarity search. In: Proceedings of the 33rd international conference on very large data bases. VLDB Endowment pp 950–961
32. Manjunath BS, Ma WY (1996) Texture features for browsing and retrieval of image data. IEEE Trans Pattern Anal Mach Intell 18(8):837–842
33. Manning CD, Raghavan P, Schütze H (2008) Introduction to information retrieval. Cambridge University Press, New York, pp 1085–1092

34. Miyagawa A, Kiyoki Y, Miyahara T, Kitagawa T (2000) A fast semantic associative search algorithm for image databases. Transactions 41:1–10
35. Moise D, Shestakov D, Gudmundsson G, Amsaleg L (2013) Indexing and searching 100 m images with map-reduce. In: Proceedings of the 3rd ACM conference on international conference on multimedia retrieval. ACM, pp 17–24
36. Navarro G, Puglisi SJ (2010) Dual-sorted inverted lists. In: International symposium on string processing and information retrieval. Springer, pp 309–321
37. Omohundro SM (1989) Five balltree construction algorithms. International Computer Science Institute, Berkeley
38. Ooi BC, McDonell KJ, Sacks-Davis R (1987) Spatial kd-tree: an indexing mechanism for spatial databases. In: IEEE COMPSAC, vol 87. p 85
39. Persin M, Zobel J, Sacks-Davis R (1996) Filtered document retrieval with frequency-sorted indexes. JASIS 47(10):749–764
40. Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2007) Object retrieval with large vocabularies and fast spatial matching. In: IEEE conference on computer vision and pattern recognition, 2007. CVPR'07. IEEE, pp 1–8
41. Sivic J, Zisserman A (2003) Video google: a text retrieval approach to object matching in videos. In: . Proceedings of the ninth IEEE international conference on computer vision, 2003. IEEE, pp 1470–1477
42. Spink A, Jansen BJ (2006) Web search: public searching of the Web, vol 6. Springer, Berlin
43. Squire DM, Müller W, Müller H, Raki J (1999) Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In: Pattern recognition letters. Citeseer, pp 143–149
44. Yan H, Ding S, Suel T (2009) Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th international conference on World wide web. ACM, pp 401–410
45. Zhang H, Zha ZJ, Yang Y, Yan S, Gao Y, Chua TS (2014) Attribute-augmented semantic hierarchy: towards a unified framework for content-based image retrieval. ACM Trans Multimed Comput Commun Appl (TOMM) 11(1):21
46. Zheng YT, Zhao M, Song Y, Adam H, Buddemeier U, Bissacco A, Brucher F, Chua TS, Neven H (2009) Tour the world: building a web-scale landmark recognition engine. In: IEEE conference on computer vision and pattern recognition, 2009. CVPR. IEEE, pp 1085–1092