VideoMec: A Metadata-Enhanced Crowdsourcing System for Mobile Videos

Yibo Wu
The Pennsylvania State University
111 IST Building
University Park, Pennsylvania 16802
yxw185@cse.psu.edu

Guohong Cao
The Pennsylvania State University
111 IST Building
University Park, Pennsylvania 16802
gcao@cse.psu.edu

ABSTRACT

The exponential growth of mobile videos has enabled a variety of video crowdsourcing applications. However, existing crowdsourcing approaches require all video files to be uploaded, wasting a large amount of bandwidth since not all crowdsourced videos are useful. Moreover, it is difficult for applications to find desired videos based on user-generated annotations, which can be inaccurate or miss important information. To address these issues, we present *VideoMec*, a video crowdsourcing system that automatically generates video descriptions based on various geographical and geometrical information, called *metadata*, from multiple embedded sensors in off-the-shelf mobile devices. With VideoMec, only a small amount of metadata needs to be uploaded to the server, hence reducing the bandwidth and energy consumption of mobile devices. Based on the uploaded metadata, VideoMec supports comprehensive queries for applications to find and fetch desired videos. For time-sensitive applications, it may not be possible to upload all desired videos in time due to limited wireless bandwidth and large video files. Thus, we formalize two optimization problems and propose efficient algorithms to select the most important videos to upload under bandwidth and time constraints. We have implemented a prototype of VideoMec, evaluated its performance, and demonstrated its effectiveness based on real experiments.

CCS CONCEPTS

•Networks →Mobile networks; •Human-centered computing →Ubiquitous and mobile computing systems and tools; •Information systems →*Multimedia databases;*

KEYWORDS

mobile video, metadata, crowdsourcing, video query, video selection

ACM Reference format:
Yibo Wu and Guohong Cao. 2017. VideoMec: A Metadata-Enhanced Crowdsourcing System for Mobile Videos. In *Proceedings of The 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, Pittsburgh, PA USA, April 2017 (IPSN 2017),* 12 pages.
DOI: 10.475/123_4

IPSN 2017, Pittsburgh, PA USA

DOI: 10.475/123_4

1 INTRODUCTION

Video technology has improved dramatically in the past decade. Large, heavy digital cameras and camcorders have been replaced with smaller, lighter smartphones capable of shooting Ultra High Definition (UHD) videos. In the near future, a variety of wearable devices such as smartglasses, smartwatches, and smart jewelry will make video recording more user-friendly. Taking a video may only require users to press a button, rather than taking their smartphones out of pockets and performing multiple touch screen actions. Through smartphones or wearable devices, the convenience of video recording is expected to significantly increase the number of mobile videos recorded and shared over the Internet.

The videos, if managed properly, could enable many useful and critical applications. For example, terrorist attacks and gun violence have been increasingly common around the world. In such incidents, authorities often identify the suspects from surveillance videos. If mobile videos taken around the time and location of the incidents were collected, they could have served as additional resources for identifying and locating the suspects, especially in places not covered by security cameras. In fact, officers investigating 2016 Brussels bombings were unable to trace a suspect along his escape route due to the absence of security cameras, and they had to ask people who inadvertently filmed or photographed the suspect to contact the police [2]. Besides crime investigation, mobile videos can be used in many other applications, such as locating a missing child or investigating traffic accidents.

To enable the above applications, one possible approach is to ask mobile users to upload all videos to the cloud or cloudlets, and then use content-based techniques to enable user search [34]. This approach, however, is extremely resource hungry. Uploading all videos consumes lots of bandwidth and energy on mobile devices, while storing and processing videos consumes a large amount of disk space and processing power on the server side. To make things worse, only a small part of the uploaded videos may be useful to the applications. Thus, the resources spent on uploading, storing, and processing all other videos are wasted.

Due to the aforementioned issues, a better approach is to let mobile devices upload high level video descriptions initially, and only ask for actual video files in response to user queries. However, existing geo-tagging features on mobile devices only tag videos with coarse location information, which is not sufficient to reveal the view of the camera. Even at the same location, cameras facing different directions will have different views. Besides location, existing systems (e.g., YouTube) rely heavily on user-generated annotations. Tagging each video manually is not convenient and

discourages public participation. More importantly, these annotations may be inaccurate or miss important information. For example, when a tourist inadvertently films a suspect, the video may be tagged as travel related, nothing to do with the suspect. Thus, it is difficult for applications to find the videos they need simply based on annotations.

To address these problems, we propose to automatically generate video descriptions based on various geographical and geometrical information, called *metadata*, from multiple embedded sensors in off-the-shelf mobile devices. Metadata includes the time of the video, the location and orientation of the camera, and other camera parameters such as field of view and focal length. Metadata can be used to infer when, where, and how the video is taken. Thus, when a crime happens at certain time and location, every video taken around that time and location can be found and used to search for the suspect, regardless of whether mobile users are aware of the suspect or not.

Overview of the proposed system. We design and implement *VideoMec*, a system that organizes and indexes all videos taken by mobile devices based on the idea of metadata, which supports efficient queries for applications to find and fetch the requested videos. VideoMec consists of two components, a VideoMec server and a VideoMec app (see Fig. 1). When a video is recorded using the VideoMec app on a mobile device, its metadata is automatically generated and uploaded to the VideoMec server. Since metadata is very small, the uploading does not consume much bandwidth and energy. In the VideoMec server, the uploaded metadata is organized and indexed in a database. Then, the application can query the database to find the desired videos, by providing information such as when and where the event happened, and from which angle or distance the event should be captured. If the video description also includes user-generated annotations, the application can provide extra keywords in the query, and the returned videos should either match the keywords or satisfy the metadata requirements. After processing the query, the VideoMec server returns the matching videos if they have been uploaded; otherwise, the server notifies the corresponding mobile devices, which automatically upload the videos and get paid [11, 22, 33] (how users will be paid is out of the scope of this paper).

When uploading videos, it is possible that mobile devices cannot upload all requested videos in time due to limited wireless bandwidth and large video files. For example, the latest smartphones can shoot UHD videos at about 40 Mbps bitrate, and then uploading one-minute UHD video would take 40 minutes for a 1 Mbps cellular connection. Since some applications are time-sensitive, requiring videos to be available within a short time, it is important to decide which videos are more valuable to the application based on the metadata, and then upload them first. To achieve this goal, we design an *upload decision engine* that decides which videos (possibly a short clip in a long video) should be uploaded given bandwidth and time constraints. The decision engine considers two typical application scenarios, one prioritizing videos that provide good time coverage, and the other prioritizing videos at the right locations.

The contributions of this paper are as follows. First, we design VideoMec, a metadata-enhanced crowdsourcing system for mobile videos. By using the uploaded metadata to serve queries and only uploading queried video files, the proposed crowdsourcing system is capable of handling a very large number of mobile videos while reducing the bandwidth and energy consumption of mobile devices. Second, VideoMec supports comprehensive queries including time, location, angle, and distance information, for applications to find desired videos quickly and accurately from the distributed video repository. Third, VideoMec's upload decision engine selects the most valuable videos to upload when bandwidth and time resources are limited. Finally, we have implemented a prototype of VideoMec, evaluated its performance and demonstrated its effectiveness based on real experiments.

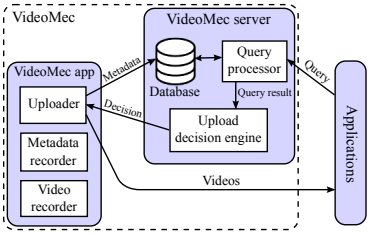


Figure 1: The VideoMec system.

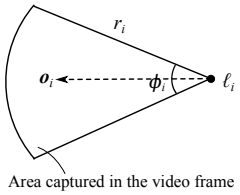


Figure 2: Metadata at timestamp t_i.

The rest of the paper is organized as follows. Section 2 introduces metadata and its storage. Section 3 describes the queries supported by VideoMec and details how those queries are processed. The upload decision engine is presented in Section 4. Section 5 evaluates the performance of VideoMec. Section 6 reviews related work, Section 7 discusses future work, and Section 8 concludes the paper.

2 METADATA AND ITS STORAGE

2.1 Metadata

A mobile video can be characterized by its metadata; i.e., when, where and how the video is taken. The metadata includes the start and end time of a video, denoted as t_s and t_e, respectively. Between t_s and t_e, we define a series of timestamps $t_0, t_1, t_2, \ldots$ such that $t_0 = t_s$ and $t_{i+1} - t_i = 1$ second for all $i = 0, 1, 2, \cdots$. Then each timestamp t_i is associated with four parameters $(\ell_i, \boldsymbol{o}_i, \phi_i, r_i)$ which jointly determine an area on the Earth's surface that is viewable in the current video frame.

As shown in Fig. 2, location ℓ_i (sometimes referred to as *video location*) is the geographic coordinates of the camera, including longitude x_i and latitude y_i. Orientation $\boldsymbol{o}_i$ is the direction to which the camera is facing. It is a vector coming from the camera aperture and perpendicular to the image plane. Field of view ϕ_i specifies how wide the camera can see. Objects outside the field of view will not appear in the video. Coverage range r_i specifies how far the camera can see. It is the distance beyond which objects are no longer clearly recognizable in the video. Details of obtaining those four parameters on mobile devices will be presented in Section 5.

Besides the above metadata, VideoMec uses frame rate and resolution available in the video header as video quality metrics. It also uses file size as a cost metric, since a larger video requires more network resources to upload.

2.2 Metadata Storage

When a mobile video is recorded, its metadata is automatically generated and uploaded to the VideoMec server. The uploaded metadata can be stored in a relational database for future queries. However, as mobile videos continue to grow exponentially, there should be efficient ways to index and query metadata.

We propose to use *R*-tree* [7] to index the metadata. R*-tree is a tree-based data structure designed for efficient indexing of spatial information such as geographical coordinates. It has been commonly used in geographical information systems (GIS) to store locations of objects such as gas stations or shapes of objects such as roads, and then find answers quickly to queries such as "find the nearest gas station" or "find all road segments within 5 miles of my location". Although R*-tree has been used in GIS, it has never been applied to video query and selection, and video metadata cannot be directly stored in R*-tree.

To use R*-tree, we convert the raw metadata of a video to the *minimum bounding box* of the video in a 3D space, which can then be stored in R*-tree for efficient query and selection. More specifically, consider the 3D space formed by the two dimensional plane of the Earth's surface[1] (called x-y plane) and the third dimension of time (called t-axis). Each timestamp t_i and its corresponding location $\ell_i = (x_i, y_i)$ maps to a point (t_i, x_i, y_i) in the t-x-y space. As time goes from t_s to t_e, point (t_i, x_i, y_i) moves in the t-x-y space and forms a trace. The minimum bounding box of the trace is defined as $(t_{min}, t_{max}, x_{min}, x_{max}, y_{min}, y_{max})$, where

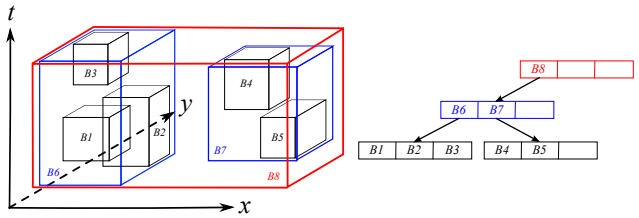
$$t_{min} = t_s, \qquad t_{max} = t_e,$$
$$x_{min} = \min_i x_i, \qquad x_{max} = \max_i x_i,$$
$$y_{min} = \min_i y_i, \qquad y_{max} = \max_i y_i.$$

This minimum bounding box represents the spatiotemporal boundary of the video, and it is stored in R*-tree.

Fig. 3 shows an example of R*-tree containing five videos. The left figure depicts the minimum bounding box of each video as B_1 through B_5. The key idea of R*-tree is to group nearby objects and represent them with their minimum bounding box in the next higher level of the tree. In this example, the left three boxes, B_1, B_2, B_3, are close to each other. Hence, they are grouped together and their minimum bounding box, B_6, is generated. Similarly, the right two boxes B_4 and B_5 are grouped and their minimum bounding box B_7 is generated. At a higher level, B_6 and B_7 are further grouped and their minimum bounding box, B_8, is generated.

The right figure shows the structure of the corresponding R*-tree. The leaf nodes store the minimum bounding box of each video, and the nodes at higher levels store larger bounding boxes which aggregate multiple videos. Based on this tree structure, consider a query to "find all videos whose minimum bounding boxes intersect with a given box", where two boxes intersect with each other if they contain at least one common point. If the given box does not intersect with a larger bounding box, e.g., B_7, it also cannot intersect with any of the contained boxes, i.e., B_4 or B_5. Therefore, the entire subtree rooted at B_7 can be skipped in the search. As R*-tree has mechanisms to maintain its balance after insertion

[1] The Earth's surface is not flat. Here we use a plane instead of a sphere for the convenience of illustration.



(a) Minimum bounding boxes (b) Tree structure

Figure 3: An example of a R*-tree containing five videos.

or deletion, it guarantees a search time of $O(\log n)$ for the above query, where n is the total number of videos in the R*-tree.

3 QUERY DESIGN AND PROCESSING

In VideoMec, an application can query the server to retrieve videos of interest. In this section, we present the supported queries and describe how those queries are processed.

3.1 Query Design

An application typically searches for videos related to a particular event. For example, a police officer may search for videos related to a traffic accident and seek evidence to determine the responsible party. Therefore, the approximate time duration and location of the queried event should be included in the query. The location of the queried event is denoted as ℓ_e. The query may also contain angle, distance, or quality requirements as follows. The query may include one or more angles $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots$ from which the event should be captured, in order to avoid obstructions (e.g., buildings and trees) blocking the line of sight. The query can also specify the distance d between video location and event location, based on how large the interested objects would appear in the video. Since the required angle or distance may not be met completely, a deviation value a_{dev} or d_{dev} is given to indicate how much deviation is allowed from the required value. Finally, a minimum acceptable frame rate or resolution can be provided to filter out videos of low quality. Fig. 4(a) shows an example of query with all the requirements specified.

3.2 Query Processing

Queries are processed using the *filter-refine paradigm*. The filter step leverages R*-tree to quickly filter out most videos that are irrelevant to the query. Then the refinement step checks the metadata of each remaining video to determine whether it matches the query or not. Details of the two steps are elaborated below.

3.2.1 Filter Step. We derive the necessary conditions that a video must satisfy in order to match the query. The conditions can be easily checked using R*-tree, and thus videos not satisfying these conditions can be quickly found and filtered out.

Consider the query shown in Fig. 4(a). We redraw the query in Fig. 4(b) and consider a video frame located at ℓ_i. For the video frame to capture the event, the distance between the video location ℓ_i and the event location ℓ_e must be within the coverage range of the camera, i.e., $\left\|\overrightarrow{\ell_i \ell_e}\right\| \leq r_i$. However, this condition cannot be

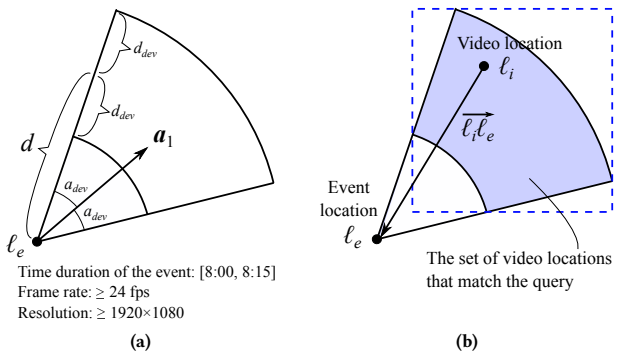


Figure 4: An example query. (a) All the requirements specified in the query. Note that there is only one required angle. (b) The set of video locations matching the query. For simplicity, assume that $d + d_{dev} \le R$.

efficiently checked using R*-tree because R*-tree only contains information about video locations while this condition also involves the coverage range. Thus, we replace r_i with a predefined constant R representing the upper bound of all r_i and obtain a necessary condition that can be efficiently checked using R*-tree:

$$\left\|\overrightarrow{\ell_i\ell_e}\right\| \le R. \tag{1}$$

When a required distance d and an allowed deviation d_{dev} are given, the distance between the video location and the event location should also satisfy

$$\left|\left\|\overrightarrow{\ell_i\ell_e}\right\| - d\right| \le d_{dev}. \tag{2}$$

If there are required angles $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots$ and an allowed deviation a_{dev}, the video frame should capture the event from an angle close to one of the required angles:

$$\min_j \angle\left(\overrightarrow{\ell_e\ell_i}, \boldsymbol{a}_j\right) \le a_{dev}. \tag{3}$$

For the video frame to match the query, its location ℓ_i must satisfy inequalities (1), (2), and (3) simultaneously[2]. On the x-y plane, the set of all video locations that satisfy the three inequalities form a closed area, as the colored annular sector shown in Fig. 4(b). To simplify the shape of that area, we find its minimum bounding rectangle shown as the dashed rectangle in Fig. 4(b). The minimum bounding rectangle and the time duration of the event together define a minimum bounding box in the t-x-y space, referred to as *query box*. For a given query, VideoMec derives its query box and uses R*-tree to find all videos whose minimum bounding boxes intersect with the query box. Then only the returned videos may match the query, and they will be further evaluated in the refinement step.

3.2.2 Refinement Step. In this step, the metadata of the videos returned from the filter step is further checked to generate the final query result.

First, if there are any video quality requirements in the query such as the minimum frame rate or the minimum resolution, videos not satisfying the quality requirements will be discarded. Then for each remaining video, since the metadata can be different for different timestamps, it is necessary to check the metadata at each timestamp to see which part of the video matches the query and which part does not. Specifically, we check whether the following four conditions hold.

(1) Timestamp t_i is within the time duration of the queried event.
(2) The distance between the video location and the event location is within the coverage range of the camera, i.e., $\left\|\overrightarrow{\ell_i\ell_e}\right\| \le r_i$.
(3) Orientation $\boldsymbol{o}_i$ is appropriate so that the event location is within the field of view of the camera, i.e., $\angle\left(\overrightarrow{\ell_i\ell_e}, \boldsymbol{o}_i\right) \le \phi_i/2$.
(4) If distance or angle is specified in the query, inequalities (2) or (3) should be satisfied.

A video frame matches the query if all four conditions hold, which means that the queried event is captured in the video frame (2nd and 3rd conditions) at the required time (1st condition) and from the required angle and distance (4th condition). When multiple video frames from the same video with consecutive timestamps all match the query, the video clip containing these frames will be returned as the output. For example, if the frames at timestamps t_2, t_3, t_4 match the query but those at t_0, t_1 and $t_5, t_6, \ldots$ do not, the video clip $[t_2, t_4]$ will be returned as an output. After all videos obtained from the filter step are checked, a set of video clips is returned as the final query result.

4 UPLOAD DECISION ENGINE

Videos returned from query processing should be uploaded by mobile devices. When uploading videos, it is possible that mobile devices cannot upload all requested videos in time due to limited wireless bandwidth and large video files. The upload decision engine decides which videos (possibly a short clip in a long video) should be uploaded given bandwidth and time constraints. Based on two typical application scenarios, we propose two different strategies.

4.1 Time-Based Strategy

The Time-Based Strategy (TBS) is motivated by applications which require uploaded videos to cover a certain period of time. Since multiple devices can take videos at the same time, uploading a video whose time interval is already covered by other videos does not increase the total time coverage. Thus, it is a challenge to select videos with the most time coverage under bandwidth and time constraints.

4.1.1 TBS Problem. Given a set of videos, we consider the problem of selecting a subset of them such that every mobile device can upload its selected videos within a time limit and the total time coverage of the selected videos is maximized. When selecting videos, it is sometimes necessary to cut a long video into smaller pieces and only upload part of the video due to time constraints. Hence, we divide a video into *segments* of fixed length L, and decide whether each segment should be selected. Having smaller L

[2]Depending on if there is any specified distance or angle, inequality (2) or (3) may or may not exist.

increases the flexibility of selecting "good" segments from different videos and different devices, thus improving the overall time coverage. However, if L is too small (e.g., one second), the quality of experience suffers since the selected videos contain many small segments and may switch scenes too frequently. The choice of L will be evaluated in Section 5. Now, we formally define the problem of selecting video segments as follows.

Definition 4.1 (TBS Problem). Consider a set of devices $D_1, D_2, \ldots, D_d$, where each device D_i has an average bandwidth B_i and a set of video segments $V_{i,1}, V_{i,2}, \ldots, V_{i,n_i}$. Each segment $V_{i,j}$ has a start time $t_s(V_{i,j})$, end time $t_e(V_{i,j})$, and file size $s(V_{i,j})$. Given an upload time limit T, the TBS problem is to select a subset of segments $\mathcal{V} \subseteq \{V_{i,j} \mid (i = 1,\ldots,d) \wedge (j = 1,\ldots,n_i)\}$ such that every device can upload its segments in $\mathcal{V}$ within the time limit, i.e., $\sum_{V_{i,j}\in\mathcal{V}} s(V_{i,j}) \leq B_i T$ for every $i = 1,\ldots,d$, and the total length of $\bigcup_{V_{i,j}\in\mathcal{V}}[t_s(V_{i,j}), t_e(V_{i,j})]$ is maximized.

Theorem 4.2. *The TBS problem is NP-hard.*

Proof. We reduce a known NP-hard problem, the 0-1 knapsack problem, to the TBS problem. In the 0-1 knapsack problem, there are a set of items, each with a value and weight. Given a knapsack with a weight limit, the problem is to find a subset of items such that their total weight is no more than the weight limit and their total value is maximized.

For any instance of the 0-1 knapsack problem, we can construct an instance of the TBS problem as follows. We construct a mobile device D_1, where the network bandwidth limit B_1T is set to the weight limit of the knapsack, and the number of video segments n_1 is set to the number of items. For the j-th segment $V_{1,j}$, its length $t_e(V_{1,j}) - t_s(V_{1,j})$ is set to the value of the j-th item, and its file size $s(V_{1,j})$ is set to the weight of the j-th item. Also, we ensure that the segments do not overlap in time because a device cannot record two videos simultaneously.

A solution $\mathcal{V}$ to this instance of the TBS problem maximizes the total length of $\bigcup_{V_{1,j}\in\mathcal{V}}[t_s(V_{1,j}), t_e(V_{1,j})]$. Since the segments do not overlap, it actually maximizes $\sum_{V_{1,j}\in\mathcal{V}}[t_e(V_{1,j}) - t_s(V_{1,j})]$. When the segments are seen as items, $\mathcal{V}$ is a subset of items which satisfies the weight constraint and maximizes the total value of items. Thus, $\mathcal{V}$ is also a solution to the 0-1 knapsack problem. This completes the reduction and hence the proof. □

The proof above shows that the TBS problem is at least as hard as the knapsack problem. Unfortunately, the TBS problem is much harder than the knapsack problem, because its objective function is *non-additive*; i.e., selecting a video segment may not increase the total time coverage. The amount of increase depends on how much the segment overlaps with others.

4.1.2 TBS Algorithm. Due to the hardness of the problem, we propose an algorithm based on several heuristics. First, let $\Delta D_i = \max\{0, \sum_{j=1,\ldots,n_i} s(V_{i,j}) - B_iT\}$ denote how much video segments in device D_i cannot be uploaded due to bandwidth constraints. If $\Delta D_i = 0$, all D_i's segments can be uploaded. Then, segments in other devices which have time overlap with D_i's segments need to be updated. That is, if a segment partially or fully overlaps with D_i's segments, the overlapping part should be discarded since it is no longer useful. By doing so, the video segments in other devices becomes shorter, and thus for those devices, ΔD_i should be checked to see if it is now equal to zero. This process, referred to as *constraint checking procedure*, identifies devices for which no selection is needed.

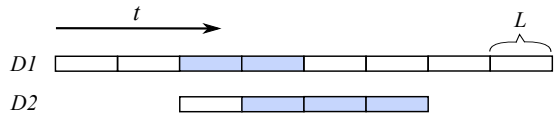


Figure 5: Suppose D_1 can upload 2 of its 8 segments, and D_2 can upload 3 of its 4 segments, so $\Delta D_1 > \Delta D_2$. If D_2 selects first, no matter what it selects, D_1 can avoid overlap and achieve a total time coverage of $5L$. If D_1 selects first, it may select two segments in the middle (shown in color). Then D_2 cannot avoid overlap and the total time coverage is at most $4L$.

After the constraint checking procedure, the remaining devices have $\Delta D_i > 0$, and we need to decide which video segments to upload for each device. We start from the device with the smallest ΔD_i. The reason is that the device with smaller ΔD_i throws away fewer segments during selection and thus is less likely to make mistakes (i.e., select segments that overlap with others). This heuristic is illustrated in Fig. 5.

For the device with the smallest ΔD_i, we select a subset of its video segments for uploading. Because segments in one device do not have time overlap, the total time coverage of those segments is equal to the sum of the length of each segment. Hence, the selection becomes a 0-1 knapsack problem where the items are video segments, the weight of a segment is its file size, and the value of a segment is its length. The knapsack problem can be solved using a polynomial time approximation scheme via dynamic programming [24]. It's time complexity is $O(n_i^3\epsilon^{-1})$, and it has an approximation ratio of $(1-\epsilon)$.

Simply applying the knapsack problem may not be the best solution. The knapsack problem values each segment by its length, but a segment overlapping with other segments in other devices is often less valuable than a "unique" segment which does not overlap with any other segments, even though the two segments have the same length. Thus, *the value of segments should be adjusted based on their time overlap.*

As an example, consider the first segment in D_1 in Fig. 6, where $L = 5$ seconds. For each second of the video, let d' denote the number of devices that have video segments in that second. Then, $d' = 0$ for the first three seconds of the video, and $d' = 2$ for the last two seconds[3]. The value of each one-second video is then decreased from 1 to $\frac{1}{1+d'}$. For example, for D_1, its value is one in the first three seconds, and its value is $\frac{1}{3}$ in the fourth and the fifth second, and then its value for the first segment is 3.66.

After value adjustment, the video segments are selected by solving the knapsack problem. Then, segments in other devices overlapping with the selected segments should be updated to remove the overlapping parts. The update may cause some devices to have

[3] For simplicity, each timestamp is rounded to its closest whole second, so that two one-second videos either completely overlap, or do not overlap at all.

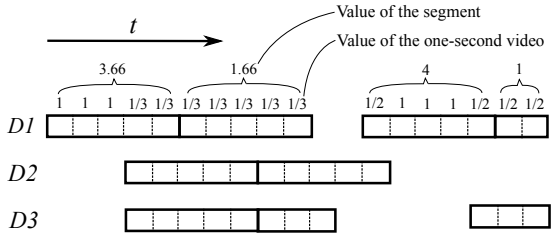


Figure 6: Value adjustments for segments in D_1.

$\Delta D_i = 0$, so a constraint checking procedure is needed and we are back to the first step. The entire process repeats until the selection is done for all devices. Then, the VideoMec server groups the selected segments that are continuous in time and in the same video, and notifies the devices which part of their videos should be uploaded. Then, the notified devices will upload the corresponding segments. Algorithm 1 shows the formal description of the TBS algorithm.

Algorithm 1 TBS algorithm

```
while selection is not done for some devices do
    while any device has ΔD_i = 0 do
        selection is done for that device;
        update segments in other devices to remove the
        overlapping parts;
    end while
    pick device D_i where i = arg min_i ΔD_i;
    for all segments V_{i,j} in D_i do
        adjust their values based on the time overlap;
    end for
    solve the 0-1 knapsack problem for D_i;
    selection is done for D_i;
    update segments in other devices to remove overlap;
end while
```

4.2 Location-Based Strategy

The Location-Based Strategy (LBS) is motivated by applications that have strict requirements on the locations of the uploaded videos. For example, a video capturing a suspect's face or a vehicle's license plate should have an appropriate angle and a close distance, so that the face or the plate number can be clearly recognized. Such information can be specified as angle and distance requirements in the query, and videos that are close to the required angle and distance should be uploaded first. Hence, we have the following optimization problem.

4.2.1 LBS Problem. Similar to the TBS problem, we consider how to select a subset of video segments such that the selected segments can be uploaded in time and their total value is maximized. Here the value of a segment is determined based on the video location, as described below.

Recall that at a given timestamp t_i, $\overrightarrow{\ell_i \ell_e}$ denotes the vector from the video location to the queried event location. Let $a_{diff} = \min_j \angle(\overrightarrow{\ell_e \ell_i}, \boldsymbol{a}_j)$ be the smallest difference between the actual viewing angle and any required angles. Then according to Section 3, only video frames satisfying $a_{diff} \le a_{dev}$ may be included in the query result. Thus, we have $a_{diff} \in [0, a_{dev}]$ for all video frames considered in the LBS problem. Similarly, let $d_{diff} = \left| \left\| \overrightarrow{\ell_i \ell_e} \right\| - d \right|$ be the difference between the actual viewing distance and the required distance. We have $d_{diff} \in [0, d_{dev}]$. Then, the value of the video frame at timestamp t_i is defined as

$$\mu \left(1 - \frac{a_{diff}}{a_{dev}}\right) + (1 - \mu) \left(1 - \frac{d_{diff}}{d_{dev}}\right),$$

where $\mu \in [0, 1]$ is a predefined constant representing the relative importance of angle against distance ($\mu = 0.5$ in our experiments). For each one-second video $[t_i, t_{i+1}]$, its value is the average value of the frame at t_i and the frame at t_{i+1}. Then, the value of a segment, denoted as $v(V_{i,j})$, is the total value of all the one-second videos. The LBS problem is formally defined as follows.

Definition 4.3 (LBS Problem). Consider a set of devices $D_1, D_2, \ldots, D_d$, where each device D_i has an average bandwidth B_i and a set of video segments $V_{i,1}, V_{i,2}, \ldots, V_{i,n_i}$. Each segment $V_{i,j}$ has a value $v(V_{i,j})$ and file size $s(V_{i,j})$. Given an upload time limit T, the LBS problem is to select a subset of segments $\mathcal{V} \subseteq \{V_{i,j} \mid (i = 1, \ldots, d) \wedge (j = 1, \ldots, n_i)\}$ such that every device can upload its segments in $\mathcal{V}$ within the time limit, i.e., $\sum_{V_{i,j} \in \mathcal{V}} s(V_{i,j}) \le B_i T$ for every $i = 1, \ldots, d$, and the total value $\sum_{V_{i,j} \in \mathcal{V}} v(V_{i,j})$ is maximized.

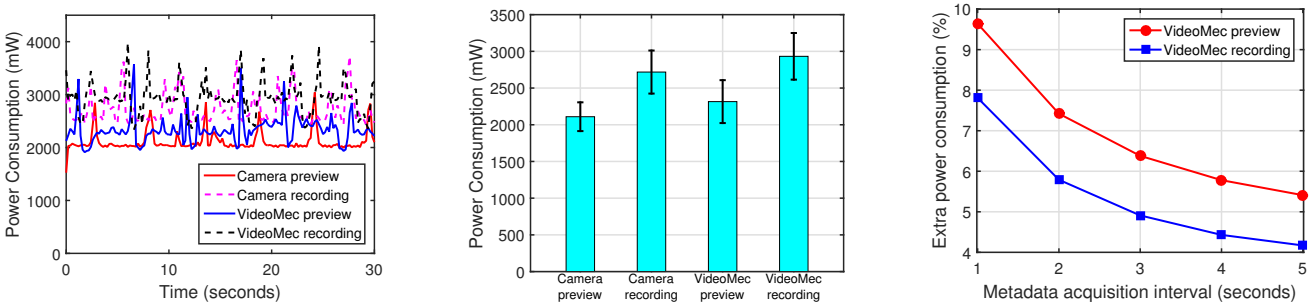
4.2.2 LBS Algorithm. Different from the TBS problem, the LBS problem is easier because its objective function is *additive*; i.e., the total value of segments equals to the sum of the value of each segment. Thus, selecting the best segments for all devices is equivalent to selecting the best segments for each device independently. Hence, the LBS algorithm solves the following problem for each device D_i.

$$\max \sum_{j \in \mathcal{J}} v(V_{i,j}),$$
$$\text{s.t.} \sum_{j \in \mathcal{J}} s(V_{i,j}) \le B_i T, \mathcal{J} \subseteq \{1, \ldots, n_i\}$$

This is a 0-1 knapsack problem where the items are video segments. As mentioned before, it can be solved by a polynomial time approximation scheme which gives a $(1 - \epsilon)$ approximation. Because the value achieved for each device is at least $(1 - \epsilon)$ times the optimal value for that device, the total value achieved by the LBS algorithm is also at least $(1 - \epsilon)$ times the optimal value of the LBS problem. Therefore, the LBS algorithm has an approximation ratio of $(1 - \epsilon)$.

5 PERFORMANCE EVALUATIONS

In this section, we present evaluation results of VideoMec. The VideoMec app was developed on Android smartphones, with the functionality of video recording, metadata recording, and video and metadata uploading. The app has been installed on four smartphones, two Samsung Galaxy S5 running Android 4.4.2 and two Motorola Nexus 6 running Android 5.1.1. The VideoMec server runs on a Dell OptiPlex 9020 desktop with Intel Core i7-4770 3.4GHz CPU.



(a) Raw measurement data

(b) Average and standard deviation

(c) Extra power vs. metadata acquisition interval

Figure 7: Power consumption of the VideoMec app vs. the Android Camera app.

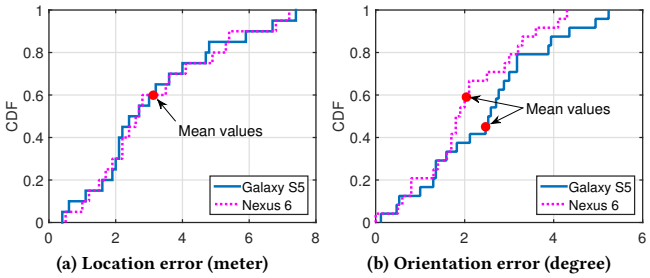
5.1 Metadata Acquisition

When a mobile user starts recording a video, a timer is started and it triggers the metadata acquisition process every second to record the current timestamp, location, orientation, field of view, and coverage range. When the video recording is finished, the timer is terminated and additional information such as frame rate, resolution, and file size is recorded.

The key issue here is how to obtain the location, orientation, field of view, and coverage range of the camera. Location is obtained from the location API of the Google Play services, which uses GPS and any other network interface such as cellular network or WiFi, to provide the best estimate of the device's true location. Certainly, other better localization techniques can be applied in the future.

Orientation is obtained from the embedded inertial measurement unit (IMU), including an accelerometer, magnetometer, and gyroscope. As a standard method documented in Android API [1], accelerometer and magnetometer can be used to obtain a *rotation matrix* of the phone, which can transform vectors in the phone coordinate system to vectors in the world coordinate system[4]. Since the orientation of the rear facing camera is $(X, Y, Z) = (0, 0, -1)$ in the phone coordinate system, the corresponding orientation in the world coordinate system can be easily obtained. However, the above method is subject to large errors due to short term vibration of the phone and ambient magnetic interference. Several techniques, most notably the use of gyroscope, can be applied to reduce such errors [35, 40]. Gyroscope measures the rate of phone rotation along the axes of the phone coordinate system. If the rate is multiplied by the period of time between two gyroscope readings, the result is the amount of rotation. Given an old rotation matrix and the amount of rotation, a new rotation matrix can be obtained. Thus, gyroscope gives another estimate of the rotation matrix, which can be combined with the estimate from accelerometer and magnetometer to obtain a more accurate result. In both [35] and [40], the authors reported orientation errors to be less than 8 degrees.

[4] In the world coordinate system, Z axis is perpendicular to the ground and points to the sky; Y is tangential to the ground and points towards the magnetic north pole; X is the vector product of Y and Z. In the phone coordinate system, Z is perpendicular to the phone screen and points outward; X is along the width of the phone and Y is along the length.



(a) Location error (meter)

(b) Orientation error (degree)

Figure 8: Measured location and orientation error.

Field of view can be calculated from three camera parameters including the imaging sensor size, focal length, and zoom level [18], all of which can be accurately obtained from the Android API [1]. Therefore, the field of view is considered to be always accurate.

The coverage range is the distance beyond which objects are no longer clearly recognizable in the video. With the pinhole camera model [18], an one-meter width object occupies $\frac{n}{2d}\cot\frac{\phi}{2}$ number of pixels horizontally in the video frame, where n is the total number of horizontal pixels, d is the distance between the camera and the object, and ϕ is the field of view of the camera. A minimum requirement on the occupied number of pixels can be given based on the recognition performance of human or computer vision algorithms. For example, to achieve certain recognition accuracy, the resolution of face images must be larger than a threshold [15]. Denoting this minimum requirement as m, we have $\frac{n}{2d}\cot\frac{\phi}{2} \geq m$, which gives $d \leq \frac{n}{2m}\cot\frac{\phi}{2}$. Therefore, the coverage range of the camera should be $\frac{n}{2m}\cot\frac{\phi}{2}$. For example, when $n = 1600, m = 40, \phi = 40^\circ$, we have a coverage range of 54.9 m.

Accuracy: We measure the location accuracy for 20 randomly picked outdoor locations on campus. For each location, we use our app to collect 10 location measurements, and the true coordinates are obtained from Google Maps. The average distance between the 10 measurements and the true coordinates is calculated, and the CDF of the average distance is shown in Fig. 8(a). As can be seen, the two smartphones we use in the experiments have similar performance, with the largest error of 7.5 m and the average error

of 3.2 m. Although GPS does not work well for indoor environments, there are many other indoor techniques to achieve good location accuracy. For example, traditional WiFi-based solutions achieve a median error of less than 10 m [26], whereas the state-of-the-art approaches using electric potential [17], RFID [29], and visible light [39] can achieve sub-meter accuracy.

We also evaluate the orientation accuracy via the following experiments. We place the phone so that its camera orientation is $0°, 15°, 30°, \ldots, 345°$ clockwise from north. For each placement, we collect 10 measurements using our app and calculate the average error against the ground truth. The CDF of the average error is shown in Fig. 8(b). As can be seen, the mean error of the two phones is around $2°$ and the maximum error is around $5°$. Compared to the usual coverage range of 50 m and the field of view of $40°$, the measured location and orientation error is relatively small. The impact of those errors on query processing will be further evaluated in Section 5.2.

Power Consumption: We use the Monsoon power monitor to measure the power consumption of a Galaxy S5 phone for both the VideoMec app and the Android Camera app[5]. For each app, we measure the power consumption of the *preview* state where camera preview is shown on the screen but no video is being recorded, and the *recording* state where a video is being recorded. During the experiments, the camera records the same scene, so that the scene displayed on the screen has the same brightness. The results are shown in Fig. 7, where Fig. 7(a) is the raw measurement data and Fig. 7(b) is the average power consumption and the standard deviation. Compared to the Android Camera app, our app consumes 9.6% more power in the preview state, and 7.8% more power in the recording state. Based on the measured power consumption, a fully charged Galaxy S5 phone (2800mAh, 3.7V) can record videos for 3.53 hours using our app, which is only 17 minutes less than using the standard Android Camera app.

The power consumption of our app can be further reduced by increasing the metadata acquisition interval. For example, location usually does not change rapidly and it can be updated every five or ten seconds. The locations between two updates can be inferred by interpolation. Fig. 7(c) shows that increasing the metadata acquisition interval to five seconds reduces the extra power consumption to 4-5%.

5.2 Effectiveness of Query Processing

In this part, we evaluate the effectiveness of query processing by comparing it with a content-based approach, which requires all video files to be uploaded and uses computer vision algorithms to find desired videos.

5.2.1 Experimental Setup. To compare VideoMec with a content-based approach, we have implemented a query processor based on optical character recognition (OCR). This query processor accepts a word as a query, and outputs a set of video frames containing the word. Specifically, for each video, it extracts one frame per second, and for each extracted frame, it runs a text localization algorithm [32] to determine which part of the frame contains text. Then the localized text is recognized by Tesseract OCR [23]. Since the recognition is subject to errors, edit distance [30] is used to determine whether the recognized text is *approximately* the same as the given word. A video frame matches the query if the minimum edit distance between the recognized text and the given word is less than a threshold.

[5] The geo-tagging feature of the Android Camera app is disabled by turning GPS and network off.

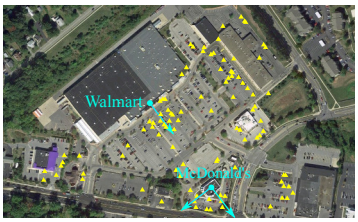


Figure 9: Satellite image of the shopping plaza. Yellow triangles are the initial locations of the 100 videos.

Figure 10: An example of correctly recognized logos.

Using VideoMec and the OCR-based query processor, we conduct the following experiments. We take 100 videos using four smartphones around a shopping plaza. The metadata (150 KB) is automatically generated and uploaded to the VideoMec server. The video files (10.7 GB) are also uploaded for the OCR-based query processor to use. Then we consider two queries: "find all videos containing the Walmart logo" and "find all videos containing the McDonald's logo". Here we consider queries on store logos because they are large, clear, and can be easily recognized by OCR.

The two queries are processed by both VideoMec's query processor and the OCR-based query processor. In addition, the ground truth is manually generated.

VideoMec. For VideoMec's query processor, we set up queries by specifying the locations, angles and allowed deviation from the specified angles. For Walmart, the event location is the position of the store logo on the building, and the specified angle is the angle from which we get the front view of the logo. For McDonald's, since it has two store logos on two sides of the building, the query location is set to be the center of the building, and there are two specified angles, one for each logo. These setups are shown as blue circles and arrows in Fig. 9. The allowed deviation from the specified angles is $45°$, and other requirements such as time and distance are left unspecified.

OCR. For the OCR-based query processor, the queries are simply two words, "Walmart" and "McDonalds". An example of correctly recognized logos are shown in Fig. 10.

Ground truth. Ground truth is obtained by visually checking each extracted frame to see whether the logo appears clearly and completely.

5.2.2 Results. By comparing the query results of VideoMec and OCR with the ground truth, we obtain several confusion matrices as shown in Table 1. Each number in the table represents the number of frames in the corresponding category. False judgments (i.e., false positives and false negatives) of VideoMec are mainly caused by large location and orientation error, which only happens occasionally. OCR, on the other hand, has more false judgments due to

(a) Walmart

		VideoMec	
	Logo?	Yes	No
Truth	Yes	492	26
	No	5	2530

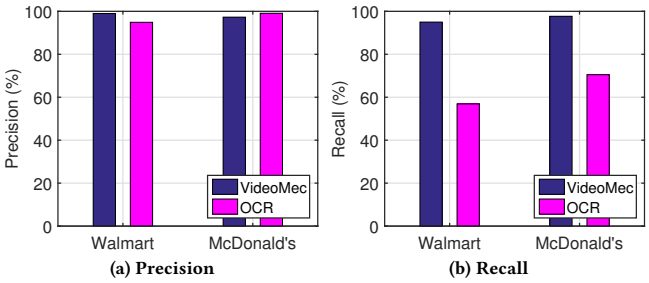
		OCR	
	Logo?	Yes	No
Truth	Yes	295	223
	No	16	2519

(b) McDonald's

		VideoMec	
	Logo?	Yes	No
Truth	Yes	295	7
	No	8	2743

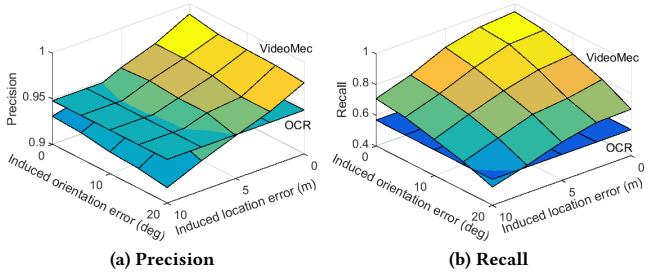
		OCR	
	Logo?	Yes	No
Truth	Yes	213	89
	No	2	2749

Table 1: Confusion matrices of the query results. Each number represents the number of frames in the corresponding category.



(a) Precision **(b) Recall**

Figure 11: Precision and recall of the query results. Precision = true positive/(true positive + false positive). Recall = true positive/(true positive + false negative).



(a) Precision **(b) Recall**

Figure 12: Precision and recall of the Walmart query vs. induced location and orientation error.

the imperfect text localization and recognition. Based on the confusion matrices, we calculate the precision and recall of queries for VideoMec and OCR, as shown in Fig. 11. Both VideoMec and OCR have very high precision ($\geq$ 95%), which means that among the frames they report as containing the logos, most indeed contain the logos. Although the recall of VideoMec is still high ($\geq$ 95%), the recall of OCR drops significantly, which means that OCR often

	VideoMec		OCR	
Preparation	Upload metadata Build database	0.24s 0.022s	Upload videos	730s
Processing	Processing Upload videos	0.004s 139s	Processing	3466s

Table 2: Query processing time of the Walmart query. Each upload time is the longest time it took for any of the four smartphones to upload the corresponding data through WiFi.

fails to report frames that actually contain the logos. This is because OCR has difficulty of locating and recognizing text viewed from the side; i.e., when there is a large difference between the actual viewing angle and the specified angle.

We further evaluate the impact of location and orientation error on VideoMec's query processing performance. The obtained metadata already contains some errors, but these errors cannot be controlled. Hence, we induce additional errors into the metadata and evaluate the sensitivity of precision and recall to errors in Fig. 12. For reference, the precision and recall of OCR are also drawn in the figure. They are not affected by metadata errors and thus appear as flat surfaces. As can be seen, the precision is not affected much by the induced errors. Even with 10 m location error and 20° orientation error, the precision is still around 92%. On the other hand, the recall decreases more when the induced errors become large. This is because when the metadata errors become larger, the number of false negatives increases much faster than the number of false positives. Nevertheless, the recall remains above 80% as long as the induced location error is less than 5 m and the induced orientation error is less than 10°. Compared to the recall of OCR (57%), the recall of VideoMec is almost always higher.

In addition to precision and recall, the query processing time is also compared in Table 2. For VideoMec, the metadata needs to be uploaded and stored into the database beforehand. Then the query can be processed quickly and videos matching the query are uploaded. If a time limit is given, VideoMec can run the TBS algorithm or the LBS algorithm to further bound the upload time (at the cost of uploading fewer, but more important videos). On the other hand, OCR takes much more time to prepare since it requires all videos to be uploaded, although many of them are not useful. It also takes more time to process the query because like most computer vision algorithms, OCR is computationally expensive.

5.3 Effectiveness of TBS and LBS algorithms

In this subsection, we evaluate the performance of the algorithms used in VideoMec's upload decision engine. To obtain the uploading bandwidth used in the algorithms, we collect 50 uplink throughput measurements using our smartphones at various locations. In each measurement, we upload a 1 MB video to the VideoMec server and calculate the throughput. Half of the uploading used cellular networks and half used WiFi. The median throughput measured for cellular networks is 3.3 Mbps, and the median for WiFi is 18.4 Mbps. Those measurements are used as traces to compare our algorithms with other algorithms under the same bandwidth setting.

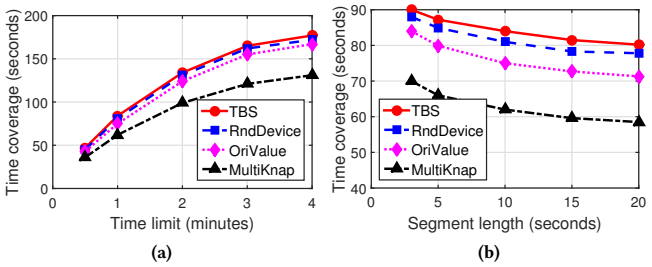


Figure 13: Comparisons between different TBS algorithms.

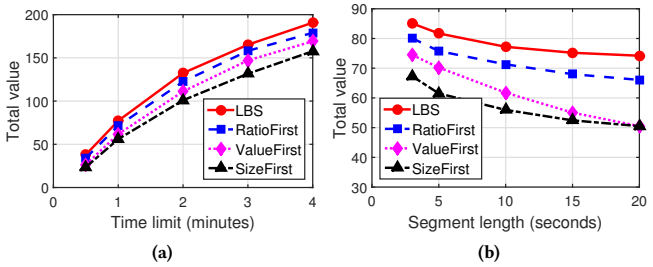


Figure 14: Comparisons between different LBS algorithms.

Five different queries are executed based on 100 collected videos. For each query, the selected videos should be uploaded to the VideoMec server. Based on the measured bandwidth, each phone will randomly pick a measurement value to be its available wireless bandwidth. Since the wireless bandwidth is limited, TBS or LBS is used to determine which part of the videos should be uploaded. For TBS, the performance is evaluated by the time coverage of the uploaded videos. For LBS, the performance is evaluated by the total value of the uploaded videos. Since there are five queries, and 10 bandwidth measurements are used for each query, we have 50 experiment runs, and each data point is averaged over those 50 runs.

5.3.1 TBS Algorithm. The TBS algorithm is compared with the following three algorithms: 1) **RndDevice:** It is based on the TBS algorithm. When deciding which device to select next, it randomly picks a device instead of choosing the one with the smallest ΔD_i. 2) **OriValue:** It is based on the TBS algorithm. Instead of value adjustment before solving the knapsack problem, it uses the original value (i.e., the length) of the video segments. 3) **MultiKnap:** It considers the TBS problem as the multiple independent knapsack problem. Each knapsack problem maximizes the time coverage for one device, while the time overlap among devices is ignored. It uses the polynomial time approximation scheme to solve each knapsack problem.

Fig. 13 (a) shows the time coverage as a function of the time limit, where the segment length L is fixed at 10 seconds. When using the polynomial time approximation scheme to solve the knapsack problem, parameter ϵ is set to 0.01. As shown in the figure, the two heuristics used in the TBS algorithm have some benefits. Choosing the device with the smallest ΔD_i instead of choosing randomly improves the time coverage by 2-4%, and using the adjusted value rather than the original value improves the coverage by 6-12%. Compared to MultiKnap, which maximizes the time coverage for each device but ignores the time overlap among devices, the TBS algorithm achieves 30-36% more coverage. For all four algorithms, the time coverage increases as the time limit increases. The increase, however, slows down gradually because with more videos uploaded, the remaining videos will be more likely to overlap with the uploaded videos, and thus contribute less coverage when being uploaded.

Fig. 13 (a) shows the time coverage as a function of the segment length, where the upload time limit T is fixed at 1 minute. As can be seen, smaller segment length increases the flexibility of selecting different parts of the video, and thus improves performance. Since switching to a different segment too frequently hurts user experience, a moderate segment length such as 10 seconds is used to balance the tradeoff between time coverage and user experience.

5.3.2 LBS Algorithm. The LBS algorithm has $(1-\epsilon)$ approximation ratio where $\epsilon > 0$ can be arbitrarily small. To evaluate its performance, we compare it with the following heuristic based algorithms: 1) **RatioFirst:** It solves the knapsack problem for each device by selecting segments from the largest value-to-size ratio to the smallest, until the upload time limit is reached. 2) **ValueFirst:** It is similar to RatioFirst, but selects segments from the largest value to the smallest. 3) **SizeFirst:** Similar to RatioFirst, but selects segments from the smallest size to the largest.

Fig. 14(a) shows the total value as a function of the time limit, where the segment length L is fixed at 10 seconds. As can be seen, the LBS algorithm performs 4-11% better than RatioFirst, 12-43% better than ValueFirst, and 21-61% better than SizeFirst. We also find that the performance difference between the LBS algorithm and the heuristic based algorithms is larger when the time limit is shorter, because with a longer upload time, good segments will always be selected even though the heuristic based algorithms do not have the optimal selection. For all four algorithms, the total value increases as the time limit increases, though the increase does not slow down as much as that in Fig. 13(a). Here the slow down occurs because some devices have selected all their segments and thus additional upload time does not provide any benefits.

Fig. 14 (b) shows the total value as a function of the segment length, where the upload time limit T is fixed at 1 minute. Similar to Fig. 13(b), the total value decreases as the segment length increases. Thus, in the LBS problem, it is also important to choose the appropriate segment length to achieve a better tradeoff between total value and user experience.

6 RELATED WORK

Video crowdsourcing. Existing works on video crowdsourcing require all video files to be uploaded to the cloud or cloudlets. For example, Chen *et al.* [9] presented a cloud leasing strategy where geo-distributed users watch live streaming videos crowdsourced from geo-distributed contributors. Simoens *et al.* [34] proposed a video crowdsourcing framework where video files are uploaded to the cloudlets (rather than the cloud). Based on the cloudlet framework, Bohez *et al.* [8] presented a live street view service. However, none of them uses fine-grained sensor data to characterize crowdsourced videos and reduce resource consumptions.

Video and sensor based applications. Applications based on mobile videos and sensor data have been proposed for various domains. For example, MoVi [6] uses various sensors to automatically detect interesting events in a social group and stitches video clips from multiple devices to produce event highlights. FOCUS [20] clusters mobile videos taken in a small area (e.g., a stadium) by camera orientation, and thus recognizes videos capturing the same event even when they have different viewing angles. Kim *et al.* [25] studied how to select frames from mobile videos, based on camera location and orientation, to generate high quality panoramic images. Different from these specific applications, we propose a general crowdsourcing framework to efficiently support video queries using metadata, which can be used in a variety of applications.

Image crowdsourcing. Image crowdsourcing has been well studied compared to video crowdsourcing. Some early work [12, 31] envisioned the possibility of using crowdsourced images and associated sensor data to construct 3D models of buildings or to improve outdoor localization. This idea was extended by Jigsaw [16], IndoorCrowd2D [10], and iMoon [13, 14] to achieve indoor navigation and floor plan reconstruction.

Several image related crowdsourcing frameworks have been proposed. For example, CrowdSearch [38] asks human workers to look at images and complete crowdsourcing tasks. In [21], Jiang *et al.* proposed to use content-based features such as color and texture to search for images on mobile devices. These content-based approaches suffer from high computational cost, and the authors had to reduce image resolution to lower the computational cost at the expense of accuracy. The authors mentioned that their solution could also be used for videos. However, when applied to videos, such computational cost will be much higher because each video has a large number of frames to be processed (even after sampling). In [35–37], metadata has been used for image crowdsourcing. They studied how to select a set of images in a single crowdsourcing task to achieve the best angle coverage on an object. In comparison, we organize and index videos to support efficient queries for all crowdsourcing tasks. Moreover, our work focuses on video while their work focuses on photos/images. Video frames cannot be simply treated as independent images because they reveal the continuous motion of objects. This brings unique challenges in the management and selection of videos, which are not addressed in previous works.

More importantly, different from these existing works, we are the first to organize and index all videos taken by mobile devices based on metadata, which supports comprehensive queries for applications to find and fetch desired videos while reducing the bandwidth and energy consumption of mobile devices.

7 FUTURE WORK

The work described here is only a first step towards scalable search and retrieval of crowdsourced videos from mobile devices. There are several important directions for future research.

Energy-Aware Uploading: The proposed TBS and LBS problems only consider the bandwidth constraint B_i of a mobile device. The total amount of data that can be uploaded (B_iT) is limited by B_i multiplied by the upload time limit T. A mobile device may have battery constraints, which limit the total amount of data to be uploaded. Such limit, denoted as E_i, can be calculated based on the energy model of the wireless interface [5, 19]. For example, if the energy model shows that under a specific network condition, uploading $1KB$ data consumes $1mJ$, then $1J$ battery energy can be used to upload $1MB$ data. In the TBS and LBS problems, we can use $\min(B_iT, E_i)$ to represent the total amount of data to be uploaded, which considers both bandwidth and energy constraints.

If bandwidth fluctuates during uploading, the solution to the TBS or LBS problem should be adjusted. The server should keep monitoring the network condition and obtaining the updated bandwidth value, which is used to solve a new instance of the TBS or LBS problem. In this new instance, videos that have been uploaded should be excluded, and the uploading process will be optimized based on the updated bandwidth value and the remaining videos.

Privacy Issues: Users may not want to release their location information and do not want the server to keep track of them. To achieve this goal, we rely on the Firebase Cloud Messaging (FCM) service [3] provided by Google. With the Firebase Library, the communication between the VideoMec server and the mobile device is through pseudo *id*s, and then the mobile device can preserve location privacy. However, if the server has some side information about the user (information it collected offline or through colluding with others), it may link the pseudo *id* to the user's real identity. Existing work on this topic [27] can be leveraged to deal with such attacks.

Besides location privacy, data privacy is also a concern. If users do not want their faces to appear in the uploaded video, techniques [4] can be applied to blur some images related to them before being uploaded. Also, some privacy leakage can be compensated by the proper design of incentive mechanisms [22, 28], which is also an important issue.

Integrating with Content-based Approaches: VideoMec and content-based approaches are different and can be complementary to each other. VideoMec supports some new applications which cannot be supported by content-based approaches. In the terrorist example mentioned in the introduction, content-based techniques may label the video captured by a tourist as travel related, and then the terrorist will never be identified without using VideoMec. On the other hand, VideoMec and content-based approaches are complementary to each other. VideoMec can be used as a filter step to quickly eliminate videos that do not match the required spatiotemporal information (or quality). Then, content-based approaches can be applied to refine the search so that videos not containing the queried object will not be uploaded. If only content-based approaches are used, retrieving all videos containing a given object will require all videos to be processed by computationally expensive algorithms, which is impossible in many cases. On the other hand, after being filtered by VideoMec, there may still be many videos matching the query, and content-based approaches can be applied to remove those videos not matching the object to save bandwidth.

Integrating with content-based approaches can also help dealing with moving objects. For a moving object, based on its initial location and time, VideoMec can be used to find videos covering that location at that time. Based on such information, VideoMec has to predict the object moving trajectory, so that more videos can be fetched. Since the prediction may be wrong, some uploaded videos

may not contain the object. With content-based approach, mobile devices can run some image recognition algorithms to check if the video contains the object before uploading to save bandwidth.

8 CONCLUSIONS

We presented VideoMec, a video crowdsourcing system that characterizes mobile videos by metadata, which can be automatically generated from multiple embedded sensors in off-the-shelf mobile devices. By using the uploaded metadata to serve queries and then only uploading the queried video files, VideoMec has much better scalability while reducing the bandwidth and energy consumption of the mobile devices. VideoMec supports comprehensive queries for applications to find videos quickly and accurately from the distributed video repository. For time-sensitive applications where not all identified videos can be uploaded in time, VideoMec selects the most important subset of videos to upload given bandwidth and time constraints. A prototype of VideoMec has been implemented, and various experiments have been conducted to demonstrate its effectiveness.

ACKNOWLEDGMENTS

We would like to thank our shepherd Dr. Tian He and the anonymous reviewers for their insightful comments and helpful suggestions. This work was supported in part by the National Science Foundation (NSF) under grant CNS-1526425 and CNS-1421578.

REFERENCES

[1] 2016. Android Developers. http://developer.android.com/index.html. (2016).
[2] 2016. New video shows alleged Brussels bomber's escape. The Washington Post. (2016).
[3] 2017. Firebase. http://firebase.google.com/. (2017).
[4] Paarijaat Aditya, Rijurekha Sen, Peter Druschel, Seong Joon Oh, Rodrigo Benenson, Mario Fritz, Bernt Schiele, Bobby Bhattacharjee, and Tong Tong Wu. 2016. I-Pic: A Platform for Privacy-Compliant Image Capture. In *ACM Mobisys*.
[5] Shravan Aras and Chris Gniady. 2016. GreenTouch: Transparent Energy Management for Cellular Data Radios. In *ACM UbiComp*.
[6] Xuan Bao and Romit Roy Choudhury. 2010. MoVi: Mobile Phone based Video Highlights via Collaborative Sensing. In *ACM MobiSys*.
[7] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD*.
[8] Steven Bohez, Jens Mostaert, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. 2014. Management of Crowdsourced First-person Video: Street View Live. In *Proc. of ACM International Conference on Mobile and Ubiquitous Multimedia*.
[9] Fei Chen, Cong Zhang, Feng Wang, and Jiangchuan Liu. 2015. Crowdsourced Live Streaming over Cloud. In *IEEE INFOCOM*.
[10] Si Chen, Muyuan Li, Kui Ren, Xinwen Fu, and Chunming Qiao. 2015. Rise of the Indoor Crowd: Reconstruction of Building Interior View via Mobile Crowdsourcing. In *ACM SenSys*.
[11] Yanjiao Chen, Baochun Li, and Qian Zhang. 2016. Incentivizing Crowdsourcing Systems with Network Effects. In *IEEE INFOCOM*.
[12] David Crandall and Noah Snavely. 2012. Modeling people and places with internet photo collections. *Commun. ACM* 55, 6 (2012).
[13] Jiang Dong, Yu Xiao, Marius Noreikis, Zhonghong Ou, and Antti Yla-Jaaski. 2015. iMoon: Using Smartphones for Image-based Indoor Navigation. In *ACM SenSys*.
[14] Jiang Dong, Yu Xiao, Zhonghong Ou, Yong Cui, and Antti Yla-Jaaski. 2016. Indoor Tracking Using Crowdsourced Maps. In *ACM/IEEE IPSN*.
[15] Clinton Fookes, Frank Lin, Vinod Chandran, and Sridha Sridharan. 2012. Evaluation of image resolution and super-resolution on face recognition performance. *Journal of Visual Communication and Image Representation* 23, 1 (2012).
[16] Ruipeng Gao, Mingmin Zhao, Tao Ye, Fan Ye, Yizhou Wang, Kaigui Bian, Tao Wang, and Xiaoming Li. 2014. Jigsaw: Indoor Floor Plan Reconstruction via Mobile Crowdsensing. In *ACM MobiCom*.
[17] Tobias Grosse-Puppendahl, Xavier Dellangnol, Christian Hatzfeld, Biying Fu, Mario Kupnik, Arjan Kuijper, Matthias R. Hastall, James Scott, and Marco Gruteser. 2016. Platypus: Indoor Localization and Identification Through Sensing of Electric Potential Changes in Human Bodies. In *ACM MobiSys*.
[18] Richard Hartley and Andrew Zisserman. 2003. *Multiple view geometry in computer vision* (2nd ed.). Cambridge University Press.
[19] Mohammad Ashraful Hoque, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao, and Sasu Tarkoma. 2015. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *Comput. Surveys* 48, 3 (2015), 39:1–39:40.
[20] Puneet Jain, Justin Manweiler, Arup Acharya, and Kirk Beaty. 2013. FOCUS: Clustering Crowdsourced Videos by Line-of-Sight. In *ACM SenSys*.
[21] Yurong Jiang, Xing Xu, Peter Terlecky, Tarek Abdelzaher, Amotz Bar-Noy, and Ramesh Govindan. 2013. MediaScope: Selective on-demand media retrieval from mobile devices. In *ACM/IEEE IPSN*.
[22] Haiming Jin, Lu Su, Houping Xiao, and Klara Nahrstedt. 2016. INCEPTION: Incentivizing Privacy-Preserving Data Aggregation for Mobile Crowd Sensing Systems. In *ACM MobiHoc*.
[23] Anthony Kay. 2007. Tesseract: an Open-Source Optical Character Recognition Engine. *Linux Journal* (2007).
[24] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems*. Springer Berlin Heidelberg.
[25] Seon Ho Kim, Ying Lu, Junyuan Shi, Abdullah Alfarrarjeh, Cyrus Shahabi, Guanfeng Wang, and Roger Zimmermann. 2015. Key Frame Selection Algorithms for Automatic Generation of Panoramic Images from Crowdsourced Geo-tagged Videos. In *Proc. of International Symposium on Web and Wireless Geographical Information Systems*.
[26] Liqun Li, Guobin Shen, Chunshui Zhao, Thomas Moscibroda, Jyh-Han Lin, and Feng Zhao. 2014. Experiencing and Handling the Diversity in Data Density and Environmental Locality in an Indoor Positioning Service. In *ACM MobiCom*.
[27] Qinghua Li and Guohong Cao. 2015. Privacy-Preserving Participatory Sensing. *IEEE Communication Magazine* (August 2015).
[28] Qinghua Li and Guohong Cao. 2016. Providing Privacy-Aware Incentives in Mobile Sensing Systems. *IEEE Transactions on Mobile Computing* (June 2016).
[29] Yunfei Ma, Xiaonan Hui, and Edwin C. Kan. 2016. 3D Real-time Indoor Localization via Broadband Nonlinear Backscatter in Passive Devices with Centimeter Precision. In *ACM MobiCom*.
[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[31] Justin Manweiler, Puneet Jain, and Romit Roy Choudhury. 2012. Satellites in Our Pockets: An Object Positioning System using Smartphones. In *ACM Mobisys*.
[32] Lukas Neumann and Jiri Matas. 2012. Real-Time Scene Text Localization and Recognition. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*.
[33] Dan Peng, Fan Wu, and Guihai Chen. 2015. Pay As How Well You Do: A Quality Based Incentive Mechanism for Crowdsensing. In *ACM MobiHoc*.
[34] Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, and Mahadev Satyanarayanan. 2013. Scalable Crowd-Sourcing of Video from Mobile Devices. In *ACM Mobisys*.
[35] Yi Wang, Wenjie Hu, Yibo Wu, and Guohong Cao. 2014. SmartPhoto: A Resource-Aware Crowdsourcing Approach for Image Sensing with Smartphones. In *ACM MobiHoc*.
[36] Yibo Wu, Yi Wang, and Guohong Cao. 2017. Photo Crowdsourcing for Area Coverage in Resource Constrained Environments. In *IEEE INFOCOM*.
[37] Yibo Wu, Yi Wang, Wenjie Hu, Xiaomei Zhang, and Guohong Cao. 2016. Resource-Aware Photo Crowdsourcing Through Disruption Tolerant Networks. In *IEEE ICDCS*.
[38] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. 2010. CrowdSearch: Exploiting Crowds for Accurate Real-time Image Search on Mobile Phones. In *ACM MobiSys*.
[39] Zhice Yang, Zeyu Wang, Jiansong Zhang, Chenyu Huang, and Qian Zhang. 2015. Wearables Can Afford: Light-weight Indoor Positioning with Visible Light. In *ACM MobiSys*.
[40] Pengfei Zhou, Mo Li, and Guobin Shen. 2014. Use It Free: Instantly Knowing Your Phone Attitude. In *ACM MobiCom*.