

Combination Shedding Schemes for Adaptive Media Workflow Execution

Lina Peng, Renwei Yu, K. Selçuk Candan, Xinxin Wang

Computer Science Department, Arizona State University, Tempe, Arizona, USA
 {lina.peng, candan, renweiyu, Xinxin.Wang.1}@asu.edu

Abstract—Complex media fusion operations can be costly in terms of the time they need to process input objects. If data arrive faster to fusion nodes than the speed with which they can consume the inputs, this will result in some input objects not being processed. In this paper, we develop load shedding mechanisms which take into consideration both data quality and expensive nature of media fusion operators. In particular, we present quality assessment models for objects and multi-stream fusion operators and we highlight that such quality assessments may impose partial orders on objects. We highlight that the most effective load control approach for fusion operators involves shedding of (not the individual input objects but) combinations of objects. Yet, identifying suitable combinations of objects in real-time will not be possible if efficient combination selection algorithms do not exist. We develop efficient combination selection schemes for scenarios with different quality assessment and target characteristics. We first develop efficient combination based load shedding when the fusion operator has unambiguously monotone semantics. We then extend this to the more general ambiguously monotone case and present experimental results that show the performance gains using quality-aware combination based load shedding strategies under the various fusion scenarios.

Index Terms—Sensor fusion, real-time systems, query processing, multimedia databases.

I. INTRODUCTION

Media processing workflows (e.g. visual sensor networks [1], MedSMan [2], and multimedia service composition [3]) implement complex multimedia processing tasks by coupling sensing, filtering, fusion, and actuation operators.

A particular challenge in real-time media workflow processing is that input data of interest may be captured faster than the speed with which nodes can process them. In such cases, nodes will need to select some of the input data for shedding in such a way that the utility losses due to the removal of objects from consideration are minimized. Therefore, to design and evaluate a load shedding scheme, it is essential to properly state the underlying *loss-of-utility* model as well as the *system constraints*. In network routing, each data packet is simply copied from the input queue of a router to its output queue; therefore, many load shedding schemes (especially those used in traditional network routing) are agnostic to the content of the data packets they consider. Most existing work on load shedding (see Section II for the related work), on the other hand, tackle the overflow problem in stream databases by aiming to maximize the throughput or at last by providing a representative sample of the results. In many media data processing applications, however, the bottleneck is the processing power of the operators and, in many cases, the throughput is constant.

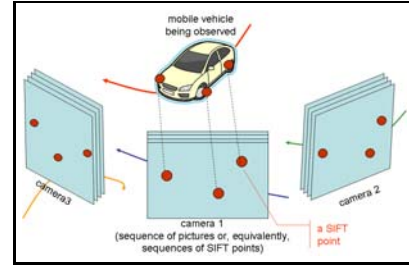


Fig. 1. A multi-camera observation system

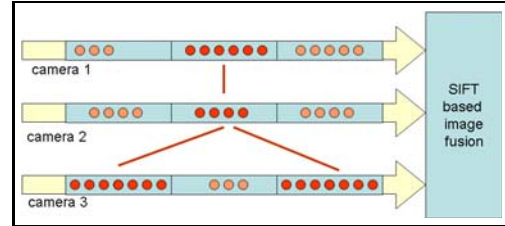


Fig. 2. Sift-based image selection: each rectangle represents a single frame and circles represent the SIFT points identified in the frame. Darker circles represent *higher* quality SIFT features, whereas lighter circles represent *lower* quality SIFT features. Taking the motion of the cameras into account, one can fuse frame combinations that are not perfectly aligned in time.

A. Quality Assessment based Shedding

In media processing, qualities of media objects depend on how they are collected and processed [4], [5]. For instance, the quality of an observation may depend on the power available on a sensor node. Similarly, the quality of a derived (or fused) observation may depend on the density [4], coverage, and power status [5] of a set of sensors covering an observation area. In this paper, we first argue that load shedding mechanisms for a media data processing environment must be cognizant of the quality assessments of the input objects. For instance, the quality of an observation depends on the power available on an adaptive sensor node. Similarly, the quality of a derived (or fused) observation may depend on the density [4], coverage, and power status [5] of a set of sensors covering an observation area.

Consider, for example, the situation in Figure 1, where three cameras on three different unmanned observation planes are used for tracking a mobile vehicle. The three cameras are streaming their individual video frames into a command-center where the frame-streams will be fused into a single combined-stream which can then be used to map the exact position and trajectory of the vehicle in the physical space. Since the three cameras themselves are independently mobile, however, the images in the individual frames need to be cali-

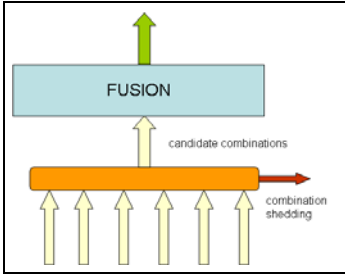


Fig. 3. The combination shedding scheme modulates the individual object-combinations that are available to the fusion operator

brated and aligned with respect to each other by determining the correspondences among salient points identified in the individual frames. Local descriptors (such as SIFT features which are invariant to image scaling, translation, and rotation, and partially invariant to illumination and projections [6], [7]) are commonly used in multi-camera calibration and image alignment applications for this very purpose. While SIFT features are somewhat robust to changes in viewing conditions and errors of the detector, however, their effectiveness still depends on various environmental parameters. Thus, since the three cameras are looking at the world from different angles, distances, and lighting conditions, there is no guarantee that all simultaneously captured frames will have sufficient detail to align them. On the other hand, leveraging (a) the correlations among the contents of the frames which are close to each other in time and (b) the knowledge about the motions of the individual cameras, these systems can fuse frames that are not perfectly synchronized with each other. Therefore, fusion candidates are frame triples that are within a small window (W); the size of the window is determined based on the speed of the cameras (Figure 2):

```
SELECT F1, F2, F3
FROM camera1 F1, camera2 F2, camera3 F3,
WHERE max(F1.time, F2.time, F3.time) -
      min(F1.time, F2.time, F3.time) <= W.
```

Obviously fusing all candidate triples (consisting of frames within W units from each other) would be extremely wasteful. Instead, the system should pick the candidates that are most likely to provide the most precise fusion result. Therefore, which frame triples will be passed to the fusion operator for processing must be chosen based on the quality assessments (number, distinctiveness, contrast, size) of their SIFT descriptors [7]. Given multiple SIFT features per frame, however, associating a single quality assessment per frame is not always affective. A statistical quality assessment of the frame, described by the mean feature quality and variance of the feature qualities, can describe the quality assessment of the overall frame better than any single value. We will revisit such statistical quality assessment models in Section IV-A.

B. Combination based Shedding

In most existing data streaming systems, overloads are handled simply by shedding individual objects within the input queues [8]–[10]. However, in the context of media fusion, multiple observation streams of a physical object can be spatially and temporally aligned and fused to augment the observation of the object. In this paper, we state that a more

effective approach is shedding (not the individual objects but) combinations of objects that are not promising (Figure 3).

Since the quality assessments of the individual objects along with the quality model of the fusion operator are all used together in selecting which object combinations to shed, combination shedding decisions can be more informed and the resulting losses of opportunities can be more constrained. Of course, when the operations themselves are cheap, yet the operators are overloaded, there is usually little to be gained from the evaluation of how *promising* input combinations are. The same resources needed for choosing the right combinations could, instead, be exploited for taking away some of the load on the overloaded operator itself. However, since media fusion operations are generally complex and expensive, there usually is a large gap between the amount of incoming data to the fusion nodes and the amount of data that can be processed by the fusion operator in real-time. In other words, when the operators (i.e., the underlying predicates) are *expensive*, we can leverage this gap between the amount of resources needed for (a) actually processing input combinations and (b) considering input combinations for elimination. In this paper, we develop combination shedding algorithms that benefit from this gap.

C. Contributions of this Paper

Based on the above observations, we propose algorithms for dealing with overloaded media fusion operators. In particular, we introduce

- quality-assessment based load shedding, suitable for sensory environments with varying qualities of data,
- realtime combination based (as opposed to input-object based) shedding for higher output qualities.

Therefore, we first provide fusion operator and object quality assessment models (Sections III and IV-A). We highlight that such quality assessments may impose partial orders on objects and identify a class of quality assessment merge functions for typical fusion operations (Section IV). Based on these semantics, we develop alternative load shedding schemes for scenarios with different quality assessment and target characteristics (Section V). In particular, unlike existing systems, we leverage this gap to pick combinations for shedding.

II. RELATED WORK

To deal with situations when the input rate exceeds the service rate of the join operators, three types of solutions have been developed in literature: object-based load shedding [8]–[13], load distribution and load diffusion strategies [14], [15], and operator re-placement [16], [17]. Load diffusion requires redundancy, while operator placement requires flexibility in the software/ hardware/ network assignments. In this paper, without having to make these assumptions, we focus on the quality-aware load shedding problem.

A. Uncertainty in Sensor Data

There have been various attempts to model and handle *imprecision* in multimedia data and sensor operations. [18] incorporates statistical models of real-world processes into a sensor-net query processing architecture. In this framework, a model is nothing but a probability density function (more

specifically a multivariate Gaussian) assigning a probability to each observation. [18] further argues that models capturing the hidden variables, such as models about sensors giving faulty values, can be learned from historical data. [19] represents imprecision by continuous probability distribution functions that indicate how the uncertain data is distributed within a given interval. The existing works mainly focus on the translation of SQL queries submitted by the users into probabilistic queries over the models and to optimize processing of these probabilistic computations relying on available knowledge about the models and their correlations. As such the contributions of these algorithms is more related to the works in the domains of fuzzy set theory, fuzzy relational databases, and probabilistic databases than the quality-aware combination shedding scheme presented in this paper.

B. Top-K Ranking and Skylines

In [20], Fagin proposed efficient top- K query execution algorithms for databases with monotonic fuzzy queries. However, random access of objects is not always available. Therefore, Fagin *et al.* [21] proposed an algorithm that performs no random access (NRA). More recently, [22] refined the NRA top- K algorithm and optimized its performance in terms of the number of object accesses, the computational cost, and the memory requirement. When data are indexed, random accesses can be avoided by intelligently accessing the join space [23]. A related problem, skyline computation of multi-dimensional data, has a wide application on multi-criteria decision support. A skyline of a set of multi-dimensional data is a set of points that are not dominated by any point in the data set in any dimension. Most published work on skyline computation assumes the multi-dimensional data set indexed by hierarchical search structure, B+ tree, R-tree, or aggregate R-tree. [24] proposes a branch-and-bound paradigm (BBS) for skyline computation on R-tree indexed multi-dimensional data. BBS progressively outputs skyline points and has been commonly cited as the best algorithm in skyline computation. A combination of top- k query and skyline query, called top- k dominating query, is also studied in [25]. Although the existing top- K and skyline algorithms provide efficient solutions to aggregate totally ordered inputs, these algorithms are not directly applicable when the inputs are only partially ordered (as frequently happens when the quality assessments are Gaussian in nature). Thus in this paper, we shall discuss how to establish rankings when incomparable inputs are possible.

C. Top-K Ranking and Skylines over Sliding Windows

Online top- k query processing over sliding windows is also considered for applications involving streaming data [26]. [26] focused on processing multiple top- k queries over a single multi-dimensional stream. An in-memory index and book-keeping structures are designed to expedite the computation of top- k query over multi-dimensional data. The index and book-keeping structures are dynamically updated for the arriving objects and expired ones. The re-computation of top- k query becomes necessary when the cardinality of the results is less than k . In this paper, we will see that incremental top- K combination selection will be in some sense similar to top- K

ranking over sliding windows. Unlike the existing work on this topic, however, combination selection may involve maintaining of candidates over partially ordered ranks. Thus, the book-keeping structures in [26] are not directly applicable.

[27] studies online skyline computation over sliding windows. In [27], an in-memory R-tree is built for (multi-dimensional) inputs and the dominance relationships between data elements are captured in the form a dominance graph. [27] proposes an incremental method for maintaining in-memory R-trees and the encoded dominance graph. In the scheme proposed in this paper, instead of an R-tree, a partial order graph (called a Hasse Diagram) is maintained to support combination shedding.

D. Top-K Ranking and Skylines over Partially ordered Data

Top- k retrieval and the related skyline problem is well studied in scenarios where data repositories are rarely updated and the objects are totally ordered in their scores. Threshold algorithm [20] for example assumes that data are sorted at each repository and the scoring function is monotone.

In [28], the scoring function is generalized to take into account a qualitative preference function which does not assign scores to objects, but specifies the preference order for a given pair of objects. The algorithm presented in [28] computes the skyline of objects associated with multiple partial scores and then progressively computes the objects that are dominated by one more object each time until the number of objects in the result set reaches k . [29] also considers skylines over partially ordered domains and, as in [28], proposes a stratification scheme to expedite skyline computation. The algorithms in [28], [29], however, assume that the data is constant (not streaming) and can be accessed in the given partial order; in this paper, however, we need incrementally maintain top- K computations over partial-orders that change as new object arrive and old ones expire.

III. FUSION OPERATOR MODEL

In this paper, without loss of generality, we model a fusion operator as follows:

Definition 3.1 (Fusion Operator (Φ)): A fusion operator, Φ , has a number of input queues, $in_qu_i, 0 \leq i \leq m$, and an output queue, out_qu . Φ picks m objects (one from each input queue) and applies a function, f_Φ , to the m -tuple of input objects and outputs the result into its output queue.

Fusion operators can have rich analysis, aggregation, and filtering semantics¹. In particular, they may perform complex media processing, information clustering, and data cleaning tasks. Thus, these operators are more similar in nature to the expensive predicates discussed in [30] than the relational stream operators considered in [9], [31].

A. Windowing Characteristics

A fusion operator has various temporal characteristics that determine its runtime properties. These include an *input selection model*, which answers the question “which possible

¹For clarity of the discussion, in this paper, we assume that the data to be fused arrive in different queues. It is trivial to extend this model when the data to be fused (e.g. clustered) is in the same queue.

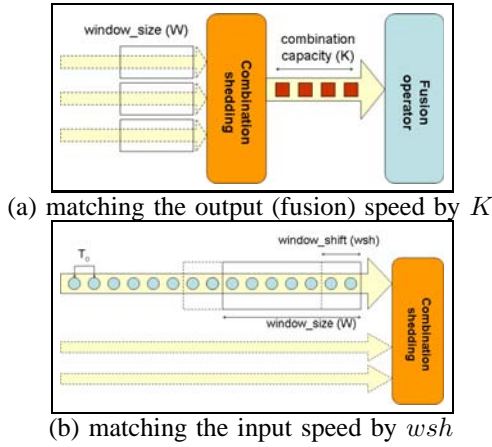


Fig. 4. Window shift (wsh) and capacity (K) parameters help match the shedding process match the input and the output speeds

candidate m -tuple in the input queue window will be selected for processing at a given point in time?” The set of objects from which Φ picks its inputs is generally governed by a time-constraint on the input objects, usually implemented through a windowing strategy [32]. In this paper, for the simplicity of discussion, we assume a simple windowing constraint where each stream has a window size, W , where the most recent W objects are treated as being valid and older objects are treated as expired. The algorithms presented in the paper, however, are not specific to this model and can be easily adapted to any windowing scheme where individual objects or object combinations can expire over time.

In the model presented above, all incoming objects are compatible with each other as long as they satisfy the given windowing constraints. This is, of course, not always the case; for example, when a multi-camera system monitors and tracks multiple objects, different objects in the environment are not compatible and their data cannot be fused. If code that can validate whether a combination is valid or not is available, extending the algorithms presented in the rest of the paper with a filter is straightforward. In many cases, however, this filter itself can be as costly as the fusion operation or may be implemented as part of the fusion operator itself. Therefore, in the rest of the paper, we ignore the object compatibility issue and assume that the fusion operator will distinguish between valid and invalid combinations.

B. Combination Selection

Given any windowing scheme on the input streams, at any given point in time, there may be more candidate combinations for processing than what the fusion operator can handle. For a combination shedding scheme to be effective, its combination selection speed has to match the object arrival speeds of the input streams as well as the combination processing (i.e., fusion) speed of on the fusion operator.

Figure 4(a) shows the output of the combination shedding process. Here, for the given $m(= 3)$ input streams and the window size W , K candidate combinations are identified and provided to the fusion operator for processing. If T_{fusion} denotes the time that it takes for the fusion operator to process a single fusion candidate, then we can state that, in the

steady state, the time, $T_{cs}(W, K, m)$ to identify the best K combinations for processing should be such that

$$T_{cs}(W, K, m) \leq T_{fusion} \times K.$$

Note that if $T_{cs}(W, 1, m) \leq T_{fusion}$, then it is possible to identify the single best combination at each iteration and simply pass this single combination to the fusion operator for processing. However, if $T_{cs}(W, 1, m) > T_{fusion}$, then it is not possible to match the speed of the T_{fusion} operator by selecting one combination at a time. In this case, it may still be possible to match the speed of the fusion operator by identifying $K > 1$ combinations per iteration, such that $T_{cs}(W, K, m) \leq T_{fusion} \times K$. As long as the K combinations identified in the following round do not overlap significantly with the K combinations identified in this round, the system is not wasting any resources and we can say that the combination shedding speed matches the fusion operator speed.

Figure 4(b) shows incoming objects on an input stream: here wsh denotes the size of the window shift at each iteration of the combination shedding process and T_o denotes the object inter-arrival time. It is easy to see that, in the steady state, if

$$T_{cs}(W, K, m) \leq T_o \times wsh,$$

the combination shedding process can be successfully applied. Note that the window shift parameter, wsh , can be varied to match any input object arrival speed: if objects are coming faster than the combination shedding algorithm can handle, then the window shift parameter, wsh , can be set to a large value to compensate the difference. However, if $wsh > W$, then some incoming objects in the stream are not considered at all, thus the process would degenerate to object shedding. Thus, in general it is desirable to have small shifts.

For a fusion operator, the window size, W , and the number, m , of streams are application specific parameters. The fusion time, T_{fusion} , depends on the underlying media processing algorithm. The window shift value, wsh , and K however can be controlled to match the speed of the combination shedding algorithm to that of the input and output speeds of the operator.

IV. QUALITY, ORDER, AND RANK

Consider, for example, an *object trajectory detector*, which visually tracks an object through cameras and identifies its trajectory. Depending on the observation/sampling rate, the quality (resolution/calibration) of the cameras, and the position of the sensor, the quality of the trajectory labels generated by detector will vary.

A. Quality Assessment Models

When the data is generated through a sensor/operator with a quantifiable quality rate (for instance a function of the available power), then a scalar-valued quality assessment may be applicable. This is similar to type-1 fuzzy predicates [33], which (unlike propositional functions which return *true* or *false*) return a membership value:

Definition 4.1 (Type-1 quality assessment): In the simplest case, the quality assessment, $qa(o)$, of a given object, o , is modeled as a constant scalar.

For example, the quality of an image reading can be quantified in terms of its resolution or the sharpness. A more

general quality assessment model would take into account the uncertainties in the quality assessments themselves. These type of predicates, where sets have grades of membership that are themselves fuzzy, are referred to as type-2 fuzzy predicates [34].

Definition 4.2 (Type-2 quality assessment): The quality assessment of a given object o is modeled as a normal distribution of qualities, $qa(o) = N_o(q_o, \xi_o)$, where q_o is the expected quality and ξ_o is its variance.

Although, the type-2 model can be more general and use different distributions, this specific model (using the normal distribution) relies on the well-known *central limit theorem*, which states that the average of the samples tends to be normally distributed, even when the distribution from which the average is computed is not normally distributed. Normal distributions are also used in different forms in data stream management context [5], [19]. Other quality assessment models are also possible. However, for our purposes, these two are enough to highlight fundamental differences in different scenarios.

B. Comparison and Ranking of the Assessments

In general, a given set, Q , of quality assessments may be either totally- or a partially-ordered:

Example 4.1 (Comparison of quality assessments): Let Q be a set of quality assessments, such that for all $qa_i \in Q$, we have $qa_i = N(q_i, \xi_i)$.

- The ordered set (Q, \preceq_c) defined as

$$(qa_i \preceq_c qa_j) \equiv_{def} \int_c^\infty qa_i(q) dq \leq \int_c^\infty qa_j(q) dq, \quad (1)$$

is totally ordered. This is because qa_i is mapped to a single scalar value which is totally ordered.

- The ordered set (Q, \preceq_p) defined as

$$(qa_i \preceq_p qa_j) \equiv_{def} q_j \geq q_i \wedge \xi_j \leq \xi_i \quad (2)$$

is partially ordered. For example, $qa_i = (0.8, 0.1)$ and $qa_j = (0.9, 0.05)$ are comparable by the definition of \preceq_p , but $qa_i = (0.8, 0.05)$ and $qa_j = (0.9, 0.1)$ are not.

The lower-bound of the set, Q , of quality assessments is

$$lbound(Q) = \{qa_i \mid (qa_i \preceq qa_j) \vee (qa_i \not\preceq qa_j), \forall qa_i, qa_j \in Q\}.$$

Given a $lbound(Q)$, $min(Q)$ is a randomly selected quality assessment in $lbound(Q)$. The upper-bound, $ubound(Q)$ and maximum quality assessment, $max(Q)$, are defined similarly.

C. Merging Quality Assessments

Each operator, Φ , has an associated quality assessment merge function, μ_Φ , describing the quality assessment of the output objects in terms of the assessments of the inputs.

Definition 4.3 (Q.A. Merge Function(μ_Φ)): A quality assessment merge function, μ_Φ , of an m -ary operator, Φ , is an m -ary function, such that given m quality assessments, qa_1, \dots, qa_m , $\mu_\Phi(qa_1, \dots, qa_m)$ is also a quality assessment.

Naturally, merge functions depend on the semantics of the fusion operation and can be arbitrarily complex. In the literature, there are a multitude of functions used for modeling

quality merge semantics for different operators [20], [33]. These include minimum and average semantics:

Definition 4.4 (Minimum Semantics): For type-1 assessments, the definition of the min semantics is straightforward:

$$\mu^{min}(Q) \equiv_{def} min(qa_1, \dots, qa_{|Q|}).$$

For (normal) type-2 quality assessments, given a set $Q = \{qa_i \mid \forall qa_i = N(q_i, \xi_i)\}$, the minimum of qa_i for Q can be defined as

$$\mu^{Nmin}(Q) \equiv_{def} N(min(q_1, \dots, q_{|Q|}), max(\xi_1, \dots, \xi_{|Q|})).$$

Note that this is an adapted version of the minimum semantics for type-1 quality assessments, where a conservative view of merging is assumed.

Definition 4.5 (Average Semantics): Average-based merge semantics is commonly used in information retrieval. Given a set Q of type-1 quality assessments, the average merge semantics can be stated as

$$\mu^{avg}(Q) \equiv_{def} \frac{\sum_{qa_i \in Q} qa_i}{|Q|}.$$

For type-2 (normal) quality assessments, the average merge semantics is defined through a convolution operator, where the convolution of $qa_i \in Q$ provides the distribution of the sum of the inputs. The convolution of two normal distributions is still a normal distribution function with mean and variance be the sum of each mean and variance respectively. Thus, if $qa_i = N(q_i, \xi_i)$ are normally distributed quality assessments in Q , then the distribution function of the average semantics for the corresponding merge function can be stated as

$$\mu^{Navg}(Q) \equiv_{def} N\left(\frac{\sum_{N(q_i, \xi_i) \in Q} q_i}{|Q|}, \frac{\sum_{N(q_i, \xi_i) \in Q} \xi_i}{|Q|^2}\right).$$

As an example, let us consider a multiple camera system for monitoring and tracking moving objects in the presence of noisy observations.

Example 4.2 (AVG Merge Func. for Trajectory Detection): Data streams from the available cameras can be fused to improve tracking quality and generate trajectory estimates using various techniques such as temporal matching, spatial matching, and Bayesian inferences [35]. Let $p\ddot{o}s_j(o)$, $1 \leq j \leq m$, denote the j^{th} camera's estimate on a given object's position in a 2D space. Let $p\ddot{o}s_{fuse}(o)$ denote the object's position obtained by fusing observations from multiple camera views. The position estimate of an object by a given camera can be represented by a vector of random variables, $[X_{est}, Y_{est}]$, where X_{est} is $N(\mu^x, \nu)$ and Y_{est} is $N(\mu^y, \nu)$. Here μ^x is the estimate of the horizontal position, μ^y is the estimate of the vertical position of the object, and ν is the variance of the position estimates based on the distance and visibility conditions. Given estimated positions, $pos_j(o) = [N(\mu_j^x, \nu_j), N(\mu_j^y, \nu_j)]$, $1 \leq j \leq m$, a fusion operator can fuse the estimates to obtain the output position $pos_{fuse}(o)$ as follows,

$$\begin{aligned}
pos_{fuse}(o) &= [X_{fuse}, Y_{fuse}] \\
X_{fuse} &= N\left(\frac{\sum_{1 \leq j \leq m} \mu_j^x}{m}, \frac{\sum_{1 \leq j \leq m} \nu_j}{m^2}\right) \\
Y_{fuse} &= N\left(\frac{\sum_{1 \leq j \leq m} \mu_j^y}{m}, \frac{\sum_{1 \leq j \leq m} \nu_j}{m^2}\right)
\end{aligned}$$

In this scenario, the quality of the result position estimate can be assessed by a scalar variance. Thus, the quality assessment $qa_{fuse}(o)$ of the position estimate is a scalar value that corresponds to the variance of $pos_{fuse}(o)$, $\frac{\sum_{1 \leq j \leq m} \nu_j}{m^2}$. Consequently, the quality assessment merge function for the multi-stream fusion operation is $\mu(\nu_1, \dots, \nu_m) = \frac{\sum_{1 \leq j \leq m} \nu_j}{m^2}$, i.e., weighted average of the input quality assessments.

D. Monotonicity

For load shedding purposes, given the underlying quality assessment model, what matters most is whether picking a better ranked quality assessment from an individual input queue would always increase the output quality assessment or not. The answer to this question depends on whether the quality assessment merge functions are monotonic or non-monotonic. In this paper, we adapt the standard definition of monotonicity as follows:

Definition 4.6 (Monotonic Merge Functions): We call an m -ary merge function, μ_Φ , monotonic (given a total or partial order, \preceq , of quality assessments) iff

$$\forall i \in \{1..m\}, qa_{1i} \preceq qa_{2i} \Rightarrow \left(\mu_\Phi(qa_{1i})\right) \preceq \left(\mu_\Phi(qa_{2i})\right)$$

It is known that the fuzzy minimum semantics ($\mu^{min}(qa_i, qa_j)$) is monotonic [20]. When applied to scalars, averaging is also monotonic; however, when applied to probability distributions, this is not always true. Let us see this with an example:

Example 4.3 (Non-Monotonicity in Merging): Let $qa_i = N(q_i, \xi_i)$, $qa_j = N(q_j, \xi_j)$ be the trajectory estimate quality assessments as introduced in Example 4.2. Then, the average-based merge function

$$\mu_\Phi(qa_i, qa_j) = N\left(\frac{qa_i + qa_j}{2}, \frac{\xi_i + \xi_j}{4}\right)$$

is non-monotonic with respect to \preceq_c as defined in Equation 1. To see this, consider that $\int_c^\infty qa_i(q) dq = \int_c^\infty N(q_i, \xi_i)(q) dq$ is known to be equal to $1 - F\left(\frac{c - q_i}{\sqrt{\xi_i}}\right)$, where the cumulative distribution function, $F(x)$, is defined as $\int_{-\infty}^x N(0, 1)(q) dq$. Therefore, given two quality assessments, qa_i and qa_j , we have

$$\mu_\Phi(qa_i, qa_j) = 1 - F\left(\frac{c - \frac{q_i + q_j}{2}}{\sqrt{\frac{\xi_i + \xi_j}{4}}}\right)$$

Now consider three quality assessments, qa_i , qa_j , and qa_k , where $qa_k \preceq_c qa_j$. It is possible to show that $\mu_\Phi(qa_i, qa_k)$ is **not** always $\preceq_c \mu_\Phi(qa_i, qa_j)$. Let, for instance, the

threshold c be 0.8. Let also (q_i, ξ_i) be $(0.6, 0)$, (q_j, ξ_j) be $(0.5, 0.4)$, and (q_k, ξ_k) be $(0.6, 0.2)$, respectively. Since $(c - q_j)/\sqrt{\xi_j} = 0.474 > (c - q_k)/\sqrt{\xi_k} = 0.447$, it follows that $qa_k \preceq_c qa_j$. However, $(c - \frac{q_i + q_j}{2})/\sqrt{\frac{\xi_i + \xi_j}{4}} = 0.79$ and $(c - \frac{q_i + q_k}{2})/\sqrt{\frac{\xi_i + \xi_k}{4}} = 0.894$. Thus, despite the fact that $qa_k \preceq_c qa_j$, $\mu_\Phi(qa_i, qa_j) \preceq_c \mu_\Phi(qa_i, qa_k)$. Therefore, μ_Φ is non-monotone.

In Example 4.3, if we change the comparison method for quality assessments from \preceq_c to \preceq_p , the average merge function becomes monotonic. However, when we are dealing with partially ordered inputs, we need to modify the definition of monotonicity to account for incomparable input objects. In particular, we define, “unambiguous monotonicity” as follows:

Definition 4.7 (Unambiguously Monotonic M.F.): An m -ary merge function, μ_Φ , is unambiguously monotonic (given a total or partial order of quality assessments), iff it is monotonic and

$$\exists i \in \{1..m\}, qa_{1i} \not\preceq_p qa_{2i} \Rightarrow \left(\mu_\Phi(qa_{1i})\right) \not\preceq_p \left(\mu_\Phi(qa_{2i})\right)$$

That is, if the inputs are incomparable, the results are also incomparable.

Example 4.4 (Unambiguous monotonicity): Let $qa_i = N(q_i, \xi_i)$ and $qa_j = N(q_j, \xi_j)$ be trajectory estimate quality assessments, with identical, constant variances $\xi_i = \xi_j = \xi$. Then, the average-based merge function

$$\mu_\Phi(qa_i, qa_j) = N\left(\frac{qa_i + qa_j}{2}, \frac{\xi}{2}\right)$$

is unambiguously monotonic with respect to \preceq_p , defined in Equation 2. This is trivially true as the inputs are all comparable.

On the other hand, when the variances of the quality assessments are not fixed, the unambiguous monotonicity does not hold. In particular, when incomparable quality assessments (with respect to \preceq_p) are averaged, the fused quality assessments may become comparable. We refer to this as “ambiguous monotonicity”:

Example 4.5 (Ambiguous monotonicity): Let $qa_i = N(q_i, \xi_i)$ and $qa_j = N(q_j, \xi_j)$ be two trajectory estimate quality assessments. Then, the average-based merge function

$$\mu_\Phi(qa_i, qa_j) = N\left(\frac{qa_i + qa_j}{2}, \frac{\xi_i + \xi_j}{4}\right)$$

is ambiguously monotonic with respect to \preceq_p as defined in Equation 2. For instance, let $qa_1 = N(0.8, 0.05)$, $qa_2 = N(0.9, 0.1)$, $qa_3 = N(0.85, 0.1)$, and $qa_4 = N(0.74, 0.2)$. Here, while $qa_4 \preceq_p qa_3$, qa_1 and qa_2 are incomparable with respect to \preceq_p . Yet, although the inputs contain a pair of incomparable objects, the following is true:

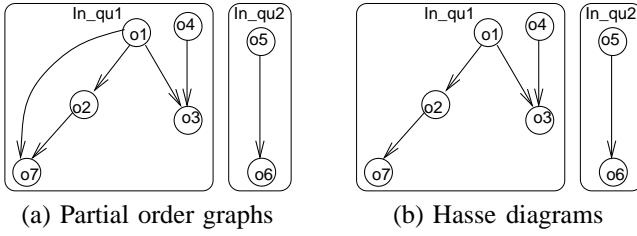


Fig. 5. Two partially ordered input queues.

$$\mu_{\Phi}(qa_2, qa_4) \preceq_p \mu_{\Phi}(qa_1, qa_3).$$

Note that, if the merge function is ambiguously monotonic, we cannot break ties while merging incomparable input objects.

V. TOP-K COMBINATION SELECTION

Overloads in queues are traditionally handled by shedding individual input objects [8]–[10]. However, as discussed in Section I-B, this approach has the disadvantage that when removed from a queue, an object cannot fuse with any other objects in the other queues. This may result in significant losses of opportunity as the removal of the objects with the lowest quality assessments does not result in an optimal shedding. *A more desirable approach would require shedding of (not the individual objects but) combinations of objects.* The challenge of course is to pick the least promising input combinations to eliminate from consideration; and perform this in realtime. The simplest approach to the problem is to enumerate all possible combinations and compute each combination’s quality assessment. If the capacity of the operator is K combinations, then we can choose the K combinations with the highest quality assessments (in total or partial order). However enumerating all possible combinations and updating them with newly arrived objects can consume a large amount of time and processing throughput. Therefore, it is essential to enumerate only the top K combinations and ignore the rest. The combinations that are ignored are not passed to the fusion operator, and thus, are effectively shed.

Monotonic quality assessment merge functions may be unambiguously or ambiguously monotone. If a merge function is ambiguously monotone, when objects with incomparable quality-assessments are considered, it is not possible to use the *domination* argument to derive predictable ranks of merged quality assessments. In this section, we discuss load shedding in this more general, **ambiguously monotonic** situations. In particular, we discuss how to establish appropriate partial orders among the combinations themselves to enable effective top- K like algorithms.

Example 5.1: Consider a fusion operator, Φ , with two input queues and fusion processing capacity of 3 tuples per second. Let the objects in one queue be o_1, o_2, o_3, o_4 , and o_7 . Let the assessment ordering of those objects be $qa(o_1) \succeq qa(o_2)$, $qa(o_1) \succeq qa(o_3)$, $qa(o_4) \succeq qa(o_3)$, $qa(o_2) \succeq qa(o_7)$, and $qa(o_1) \succeq qa(o_7)$. The objects in the second queue are o_5 and o_6 , where $qa(o_5) \succeq qa(o_6)$.

Figure 5(a) presents a graphical representation of the two input queues. For each input queue, the graph reflects the partial order among the quality assessments of the corresponding objects. An edge between two objects indicates

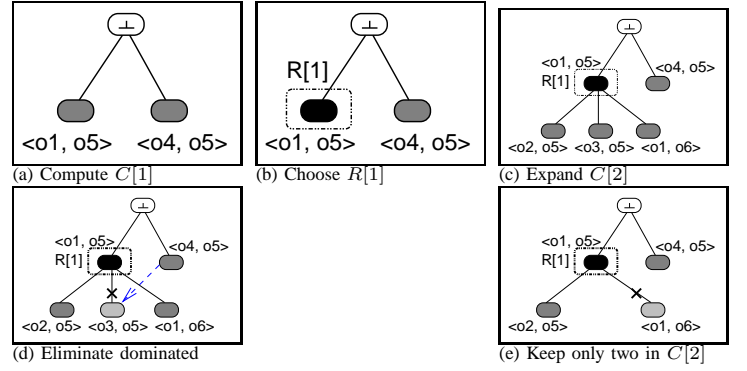


Fig. 6. Updating the candidate set C ($k = 1$).

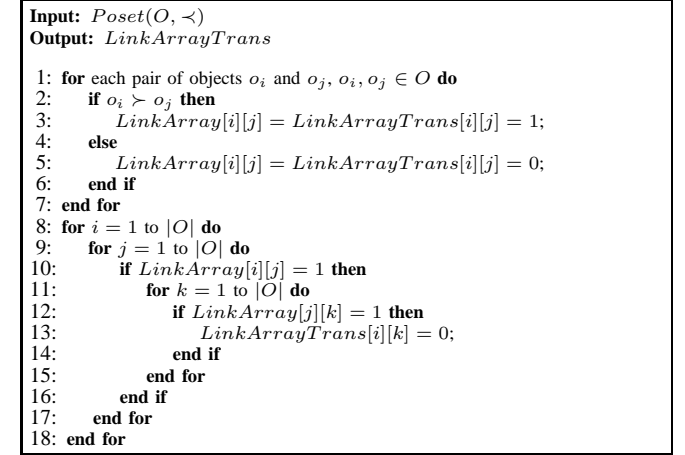


Fig. 7. Hasse diagram construction

that they are comparable in terms of their quality assessments; the direction of the edge is from the better to the worse object. In this example, since the quality assessments of o_1 and o_4 are incomparable, if the merge function is not unambiguously monotonic, then it is not possible to decide whether $\langle o_1, o_5 \rangle$ or $\langle o_4, o_5 \rangle$ will lead to a better result.

The main intuition in the above example is that, although we do not exactly know which one, we know that the best combination is either $\langle o_1, o_5 \rangle$ or $\langle o_4, o_5 \rangle$. Thus using the partial order graphs of the inputs, we can compute the top- K combinations.

A. Basic (non-incremental) Combination Selection Algorithm

Consider a fusion operator, Φ , which has m input queues, in_qu_i , $1 \leq i \leq m$. Let O_i denote the set of objects in input queue in_qu_i . Suppose that the corresponding quality assessment merge function is ambiguously monotonic. We need to handle the cases when the merge function is ambiguous due to input objects which are incomparable. The proposed algorithm (Figure 8) is discussed, step by step, next.

(1:) Initialization. Initially, the *Result* set is empty. The result counter, k , is set to 1.

(2-4:) Identification of the partial orders of the objects. This involves pairwise-wise object comparisons within each individual window. Let O be a set of objects in a given stream window. A Hasse diagram describes a partially ordered set (poset) (O, \prec) such that the edges between objects $o_i, o_j \in O$ indicate that $o_i \succ o_j$ and $\nexists o_k \in O, o_i \succ o_k \succ o_j$. Figure 7

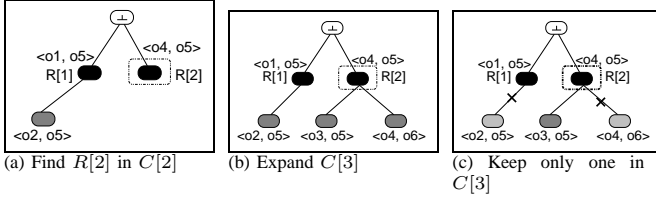
Input: Given m input streams and a fusion operator Φ .
Output: K combinations with the best quality assessment rank.

```

1:  $Result = \emptyset$ ;  $k = 1$ ;
2: for  $i = 1$  to  $m$  do
3:    $Hasse(Poset(O_i, \prec))$ ; {Procedure in Figure 7}
4: end for
5:  $C[k] \leftarrow \prod_{i=1}^m maxset_i$ ;
6: while  $C[k]$  and  $k \leq K$  do
7:   find  $c_{best} \in C[k]$  with the best ranked quality assessment;
8:    $Result \leftarrow Result \cup \{c_{best}\}$ ;
9:    $C[k+1] \leftarrow C[k] \setminus \{c_{best}\}$ ;
10:  for each  $obj'_i \in children(obj_i)$ ,  $obj_i = c_{best}[i]$ ,  $1 \leq i \leq m$  do
11:     $c' = c_{best}$ ;  $c'[i] = obj'_i$ ;
12:    if  $c'$  is not dominated by  $c \in C[k+1]$  then
13:       $C[k+1] \leftarrow C[k+1] \cup \{c'\}$ ;
14:    end if
15:  end for
16:   $k = k + 1$ ;
17: end while

```

Fig. 8. Basic combination selection algorithm

Fig. 9. Updating the candidate set C ($k = 2$).

shows the Hasse diagram construction algorithm. Given the set of objects, O , initially, a link array matrix, $LinkArray$, holds the complete ordering information for each pair of elements in a poset. $LinkArray[i][j] = 1$ if $o_i \succ o_j$, $\forall o_i, o_j \in O$; otherwise, $LinkArray[i][j] = 0$. Since a Hasse diagram takes transitivity into account and does not explicitly store the ordering information that is implied by transitivity, another link array matrix, $LinkArrayTrans$, holds the edge information; i.e., $LinkArrayTrans[i][j] = 1$ iff $\langle o_i, o_j \rangle$ is an edge in Hasse diagram. The edges that are implied by transitivity are removed: if $LinkArray[i][j] = 1$ and $LinkArray[j][k] = 1$, this implies that $LinkArrayTrans[i][k] = 0$. The computational cost of constructing a Hasse diagram is $O(|O|^3)$.

(5:) Building the candidate set $C[1]$. Due to the possibility of incomparable inputs, we cannot simply combine the first ranked objects to get the first candidate set to consider.

Let $maxset_i$ be the set of objects which have their quality assessments in $ubound(Q_i)$, where Q_i is the quality assessments of objects in in_qu_i and $ubound(Q_i)$ is defined as in Section IV-B. Graphically, this corresponds to the roots of the partial order graph established for objects in O_i . The candidate set, $C[1]$, contains the K best ranked m -tuples in the Cartesian product of all $maxset_i$ s.

Example 5.2: Reconsider Example 5.1 discussed earlier. The Cartesian product of the two sets of roots ($\{o_1, o_4\}$ and $\{o_5\}$) gives two 2-tuples, $\langle o_1, o_5 \rangle$ and $\langle o_4, o_5 \rangle$ (Figure 6(a)).

(6:) While $C[k] \neq \emptyset$ and $k \leq K$ do

(7:) Find c_{best} with the best ranked quality assessment among all the candidates in $C[k]$. In other words, if Q_C is the bag of quality assessments of objects in $C[k]$, c_{best} is an object in $C[k]$ whose assessment is equal to $max(Q_C)$.

Example 5.3: In Figure 6(b), the 2-tuple $\langle o_1, o_5 \rangle$ is marked as the best 2-tuple after comparing it with $\langle o_4, o_5 \rangle$.

Note that this does not assume that the merge function

is unambiguously monotone and the incomparable merged results are totally ordered by a ranking function. Ties between the merged quality assessments are broken arbitrarily.

(8:) Move c_{best} to result: $R[k] = c_{best}$; $Result \leftarrow Result \cup \{R[k]\}$; $C[k+1] \leftarrow C[k] - \{R[k]\}$

In other words, in this step, c_{best} is identified as the k^{th} result ($R[k]$), and it is removed from further consideration. Let us denote the object, in $R[k]$, coming from the input object set O_i as $R[k][i]$.

(9:) Update the candidate set, $C[k+1]$. Updating the candidate set $C[k+1]$ consists of three steps: (i) identification of new candidate combinations, (ii) inclusion of the new candidates in $C[k+1]$, and (iii) discarding of the combinations that are dominated by others in $C[k+1]$ or ranked worse than $K - k$ in $C[k+1]$. These are detailed below.

(10-15:) Identification and inclusion of new candidate combinations. Let us first introduce the concepts of children and descendants of objects.

Definition 5.1 (Children and Descendants): Given a set, O , of objects and an object $o \in O$, let $O^{\leq o} = \{o' | qa(o') \preceq qa(o)\}$. Then, $children(o)$, is the set of objects whose quality assessments are in $ubound(Q'_i)$, where Q'_i is the bag of assessments of the objects in $O^{\leq o}$. Intuitively, the elements of $children(o)$ are the immediate descendants of the object o in the corresponding partial order graph. Definition of $descendants(o) (\supseteq children(o))$ follows naturally.

The new candidates are discovered by replacing each input object, $R[k][i]$, of the combination $R[k]$ (one at a time) with the set of objects, $children(R[k][i])$.

Example 5.4: Figure 6(c) shows three potential candidates, $\langle o_2, o_5 \rangle$, $\langle o_3, o_5 \rangle$ and $\langle o_1, o_6 \rangle$, that can be discovered by replacing o_1 with o_2/o_3 and o_5 with o_6 , respectively. Note that $\langle o_7, o_5 \rangle$ does not need to be considered simply because object o_7 is just in $descendants(o_1)$ rather than $children(o_1)$.

Note that not all discovered candidates are necessary in $C[k+1]$. First of all, if $c = \langle obj_1, obj_2, \dots, obj_m \rangle$, then no $c' = \langle obj'_1, obj'_2, \dots, obj'_m \rangle$ is in $C[k+1]$, such that

$$(\exists_h qa(obj'_h) \prec qa(obj_h)) \wedge (\forall_{i \neq h} qa(obj'_i) \preceq qa(obj_i))$$

needs to be in $C[k+1]$.

Example 5.5: The dashed edge in Figure 6(d) shows that the combination $\langle o_4, o_5 \rangle$ dominates $\langle o_3, o_5 \rangle$ (note that, in the partial order graph in Figure 5, there is an edge from o_4 to o_3 but not vice versa; thus $qa(o_3) \prec qa(o_4)$); thus $\langle o_3, o_5 \rangle$ is discarded from consideration.

Finally, since we need to identify at most K matches and since we already identified k of them, we need to maintain at most $K - k$ candidates in $C[k+1]$. This is achieved by eliminating the extra candidates with the smaller quality assessments (ties among the merged quality assessments are broken arbitrarily). In Figure 6(e), $\langle o_1, o_6 \rangle$ is discarded (i.e., its quality assessment is the smallest) since $K = 3$.

(16:) Increment the result counter; i.e., $k = k + 1$.

After this, the while loop (lines 6-16) is repeated until the termination condition is satisfied.

Example 5.6: Figure 9(a-c) illustrates the execution of the second iteration for the running example. Note that the 2-tuple $\langle o_3, o_5 \rangle$, which was deleted from C in the first iteration, is reintroduced due to the selection of $\langle o_4, o_5 \rangle$ as $R[2]$.

B. Correctness of the Basic Algorithm

Given a fusion operator, Φ , with m input queues, $in_qu_i, 1 \leq i \leq m$, and m sets of input objects O_1, \dots, O_m , each corresponding to one input queue, if the merge function of μ_Φ is monotonic, then the following are true.

Lemma 5.1 (Children of a Combination): When a combination, $c = \langle obj_1, obj_2, \dots, obj_m \rangle$, is expanded (one obj_i at a time) with *children*(obj_i), then each resulting combination, c' , is in *children*(c).

This and the next corollary follow from the definitions of *children*(\cdot), *ubound*(\cdot), and monotonicity of the function.

Corollary 5.1 (Qualities of the Descendants): Given a combination, c , all combinations $c' \in \text{children}(c)$ are of inferior quality assessment than c . All $c', c'' \in \text{children}(c)$ are incomparable. Furthermore, for all $c''' \in \text{descendants}(c) - \text{children}(c)$, there is at least one $c' \in \text{children}(c)$ such that $qa(c''') \preceq qa(c')$.

This gives us a way to suitably expand the candidate set.

Lemma 5.2 (One-Step): Let C be a set of combinations and $r_\Phi(\cdot)$ be a ranking function. Then, if $c \in C$ is the best ranked combination in C , then

$$C' \leftarrow C - \{c\} \cup \text{children}(c),$$

contains the next best ranked combination, c' , in

$$C'' \leftarrow C - \{c\} \cup \left(\bigcup_{c' \in C} \text{descendants}(c') \right).$$

This follows from Lemma 5.1, Corollary 5.1, and the definition of ranking function. Then, we state the progress property of the algorithm as follows.

Lemma 5.3 (Progress): Given a ranking function, $r_\Phi(\cdot)$, the above algorithm ensures that at the start of the k^{th} iteration, the k^{th} best ranked m -tuple will be in $C[k]$.

The proof is by induction through the Lemma 5.2.

Theorem 5.1 (Correctness): Given a ranking function, $r_\Phi(\cdot)$, of quality assessments, the set *Result* contains the k best ranked m -tuples relative to $r_\Phi(\cdot)$ after k iterations.

This correctness result follows from Lemmas 5.3 and 5.2.

C. Complexity of the Basic Algorithm

In the general case, where $f \ll n$, this is significantly more efficient than the $O(n^m t_{eval})$ brute force algorithm:

Theorem 5.2 (Time Complexity): Given the sets, O_1, \dots, O_m , of objects, let f denote the largest mutually-incomparable subset of objects any O_i contains. Let also $n = \max\{|O_1|, \dots, |O_m|\}$. Let t_{eval} be the time needed for evaluating the merged quality assessment of a given m -tuple. Then, the algorithm runs in $O(mn^2 + mf^m \log f + K^2 m^2 f + (K^2 + mfK) \log(K + mf) + (mfK + f^m) t_{eval})$.

The algorithm uses $O(Km + m^2 f + f(\sum_{1 \leq i \leq m} |O_i|))$ runtime space.

Proof is omitted for space consideration (see the Appendix).

```

Input:  $o_{new}, H, maxset$ 
1:  $UC(o_{new}) \leftarrow \emptyset; LC(o_{new}) \leftarrow \emptyset; descendants(o_{new}) \leftarrow \emptyset;$ 
2: for  $o_i \in maxset$  do
3:   if  $o_i \prec o_{new}$  then
4:      $maxset \leftarrow maxset \setminus \{o_i\}; LC(o_{new}) \leftarrow LC(o) \cup \{o_i\};$ 
5:   else if  $o_{new} \prec o_i$  then
6:      $UC(o_{new}) \leftarrow UC(o_{new}) \cup \{o_i\};$ 
7:   else  $\{o_{new} \sim o_i\}$ 
8:      $indifferent(o_{new}) \leftarrow indifferent(o_{new}) \cup \{o_i\};$ 
9:   end if
10: end for
11:  $UpperCover(UC(o_{new}), o_{new});$ 
12: for  $o_i \in indifferent(o_{new})$  do
13:   for  $o_j \in LC(o_i)$  do
14:     if  $o_j \prec o_{new}$  then
15:        $descendants(o_{new}) \leftarrow descendants(o_{new}) \cup \{o_j\};$ 
16:     else  $\{o \sim o_j\}$ 
17:        $indifferent(o_{new}) \leftarrow indifferent(o_{new}) \cup \{o_j\};$ 
18:     end if
19:   end for
20: end for
21:  $LowerCover(descendants(o_{new}), LC(o_{new}), o_{new});$ 
22: for  $o_i \in LC(o_{new})$  do
23:    $UC(o_i) \leftarrow UC(o_i) \cup \{o_{new}\};$ 
24: end for
25: for  $o_i \in UC(o_{new})$  do
26:    $LC(o_i) \leftarrow LC(o_i) \cup \{o_{new}\};$ 
27: end for
28: if  $UC(o_{new}) = \emptyset$  then
29:    $maxset \leftarrow maxset \cup \{o_{new}\};$ 
30: end if
31:  $H \leftarrow H \cup \{o_{new}\};$ 

```

Fig. 10. *InsertObject*: inserts the arriving object to the Hasse diagram

```

Input:  $UC(o_{new}), o_{new}$ 
1: for  $o_i \in UC(o_{new})$  do
2:   for  $o_j \in LC(o_i)$  do
3:     if  $o_j \prec o_{new}$  then
4:        $UC(o_{new}) \leftarrow UC(o_{new}) \cup \{o_j\} \setminus \{o_i\};$ 
5:     else if  $o_{new} \prec o_j$  then
6:        $descendants(o_{new}) \leftarrow descendants(o_{new}) \cup \{o_j\};$ 
7:        $UC(o_j) \leftarrow UC(o_j) \setminus \{o_i\}; LC(o_i) \leftarrow LC(o_i) \setminus \{o_j\};$ 
8:     else  $\{o_{new} \sim o_j\}$ 
9:        $indifferent(o_{new}) \leftarrow indifferent(o_{new}) \cup \{o_j\}$ 
10:    end if
11:   end for
12: end for

```

Fig. 11. *UpperCover*: top-down search for objects that cover o_{new}

D. Incremental Combination Selection Algorithm

The algorithm described in Section V-A takes a set of objects for each input stream, creates a Hasse diagram, and performs top- K combination selection. A naive approach to operating on streams of objects would be to reconstruct the Hasse diagram from scratch whenever new objects arrive or expire and to re-compute the best K combinations based on the new Hasse diagrams. As we have seen in Section III, however, consecutive windows over a given stream can overlap significantly depending on the value of the window shift parameter, wsh . This means that a re-construction based scheme would simply waste a lot of resources to re-compute what has already been computed before.

A scheme which injects new objects into an existing Hasse diagram, eliminates the expired ones, and incrementally computes the impact of these additions/deletions on the set of candidate combinations made available to the fusion operator can save time by reducing the Hasse diagram and result set construction costs. This section presents an on-line solution to select of the best K combinations over frequently updated, partially ordered streaming inputs.

Incremental Maintenance of the Hasse Diagram. Let O be a set of objects and $o_i, o_j \in O$. We say that the object o_i is

```

Input:  $descendants(o_{new}), LC(o_{new}), o_{new}$ 
1: for  $o_i \in descendants(o_{new})$  do
2:   for  $o_j \in descendants(o_{new})$  do
3:     if  $o_i < o_j$  then
4:        $descendants(o_{new}) \leftarrow descendants(o_{new}) \setminus \{o_i\}$ ;
5:     end if
6:   end for
7: end for
8: for  $o_i \in descendants(o_{new})$  do
9:   for  $o_j \in LC(o_{new})$  do
10:    if  $o_i < o_j$  then
11:       $descendants(o_{new}) \leftarrow descendants(o_{new}) \setminus \{o_i\}$ ;
12:    else
13:       $LC(o_{new}) \leftarrow LC(o_{new}) \cup \{o_i\}$ ;
14:    end if
15:  end for
16: end for

```

Fig. 12. *LowerCover*: checking $descendants(o_{new})$ for elements in $LC(o_{new})$

covered by o_j ($o_i \prec_{cvr} o_j$) if $qa(o_i) < qa(o_j)$ and $qa(o_i) \preceq qa(o'_i) < qa(o_j)$ implies $o_i = o'_i$. In other words, there is no element $o'_i \in O$ such that $qa(o_i) \preceq qa(o'_i) < qa(o_j)$ (or $qa(o_i) \succeq qa(o'_i) > qa(o_j)$). The Hasse diagram corresponding to O represents a partially ordered set by associating each $o_i \in O$ with a node and each covering pair $o_i \prec_{cvr} o_j$ in O with a link from o_j to o_i . Thus, the key to incrementally maintaining the Hasse diagram is to be able to efficiently identify the set of nodes in the diagram that covers the given, o , and the set of nodes in the diagram that are covered by o . We call the former set “the upper cover”, $UC(o)$ and the later set “the lower cover” $LC(o)$, of object o . We also say that an object $o_i \in O$ is a maximal object (or a minimal object) if there is no object $o_j \in O$ such that $qa(o_j) \succ qa(o_i)$ (or $qa(o_j) < qa(o_i)$). The *maxset* (*minset*) denotes the set of objects which are maximal (minimal) in O .

Figure 10 illustrates the algorithm that incrementally includes arriving objects in the constructed Hasse diagram. The inputs to the algorithm are the new object o_{new} to be placed in the diagram, the current Hasse diagram, H , implemented as a hash table that maps each $o \in O$ to its upper cover set $UC(o)$ and lower cover set $LC(o)$, and $maxset(O)$. At the end of each iteration of the incremental construction algorithm, o_{new} is placed in the hash table corresponding to the Hasse diagram with its newly computed $UC(o_{new})$ and $LC(o_{new})$:

- Upon arrival of a new object o_{new} , *InsertObject* starts by comparing o_{new} with the objects in $maxset$.
- The maximal objects that are dominated by o_{new} are immediately placed in $LC(o_{new})$ since there does not exist other objects that dominate the maximal objects.
- The maximal objects that dominate o_{new} are temporarily placed in $UC(o_{new})$. However, we cannot immediately determine whether the maximal objects that dominate o_{new} cover o_{new} without looking at their descendants. Figure 11 illustrates the *UpperCover* algorithm that starts with the objects in $UC(o_{new})$ and explores the descendants of each object currently in $UC(o_{new})$.

- If an object $o_i \in UC(o_{new})$ has a descendant o'_i that dominates o_{new} , this implies that there exists another object $o_j \in O$ such that $qa(o_i) \succ qa(o_j) \succ o_{new}$ and o_i should be replaced by o_j as a candidate object to $UC(o_{new})$; otherwise, o_i stays in $UC(o_{new})$.

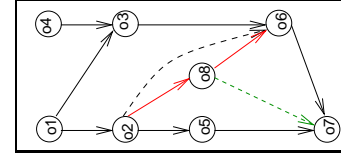


Fig. 13. The new object o_8 is inserted into an existing Hasse diagram: the existing edge between o_2 and o_6 is broken by the new object

- If an object $o_i \in UC(o_{new})$ has a descendant o_j that is dominated by o_{new} o_j is temporarily placed in $LC(o_{new})$. Note that, in this case, o_{new} is an object between o_i and o_j : $qa(o_i) \succ qa(o_{new}) \succ qa(o_j)$. Thus, the covering relation between o_i and o_j is modified at Line 7 in Figure 11.
- If an object $o_i \in UC(o_{new})$ has a descendant o_j that neither dominates, nor is dominated by o_{new} , then o_j is placed into *indifferent*(o_{new}).
- The descendants of indifferent objects can be either indifferent to o_{new} or dominated by o_{new} . Line 15 in Figure 10 places the descendants of each object $o_i \in indifferent(o_{new})$ that are dominated by o_{new} into $descendants(o_{new})$.
- The *LowerCover* procedure in Figure 12, then, examines the objects in $descendants(o_{new})$ and removes those that are dominated by another object in $descendants(o_{new})$ or an object in $LC(o_{new})$.

Example 5.7: Consider the poset $(O, <)$ with $O = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$ in Figure 13 where o_1 and o_4 are the maximal elements of the poset and the edges included in the diagram before o_8 arrives are indicated by black solid or dashed arrows. When o_8 arrives, it is compared with the objects in $maxset$. o_8 is dominated by o_1 but covered by its descendant o_2 . o_4 , o_3 , and o_5 are found indifferent to o_8 and o_6 is dominated by o_8 and placed in $descendants(o_8)$. For indifferent object o_5 , its descendant o_7 is dominated by o_8 and is placed in $descendants(o_8)$. Since o_7 is dominated by o_6 , it is removed from $descendants(o_8)$ and o_6 is covered by o_8 . The covering relation included in the diagram after o_8 arrives are indicated by red arrows. The green dashed arrow indicates an unqualified covering edge between o_8 and o_7 , which is removed. The black dashed arrow indicates a previous covering edge that was removed due to the arrival of o_8 .

When an object o_{old} expires, the *DeleteObject* procedure in Figure 14 removes o_{old} from the lower cover set of each object $o_i \in UC(o_{old})$ and the upper cover set of each object $o_j \in LC(o_{old})$. If o_{old} is a maximal object, it is also removed from $maxset$; furthermore, those objects in $LC(o_{old})$ which were only covered by o_{old} are now included in $maxset$. Note that an object $o_j \in LC(o_{old})$ can become covered by an object $o_i \in UC(o_{old})$ if there is no object $o_k \in LC(o_i)$ that dominates o_j . In this case, the new covering relation after the removal of o_{old} is also included in the new Hasse diagram.

Incremental Maintenance of the Result Set. As time moves, *ResultSet* is subject to various changes:

- some combinations are consumed by the fusion operator

```

Input:  $o_{old}, H, maxset$ 
1: if  $UC(o_{old}) = \emptyset$  then
2:    $maxset \leftarrow maxset \setminus \{o_{old}\}$ ;
3:   for  $o_i \in LC(o_{old})$  do
4:      $UC(o_i) \leftarrow UC(o_i) \setminus \{o_{old}\}$ ;
5:     if  $UC(o_i) = \emptyset$  then
6:        $maxset \leftarrow maxset \cup \{o_i\}$ ;
7:     end if
8:   end for
9: end if
10: for  $o_i \in UC(o_{old})$  do
11:    $LC(o_i) \leftarrow LC(o_i) \setminus \{o_{old}\}$ ;
12:   for  $o_j \in LC(o_{old})$  do
13:      $UC(o_j) \leftarrow UC(o_j) \setminus \{o_{old}\}$ ;
14:     if  $\neg \exists o_k \in LC(o_i)$  such that  $o_j \prec o_k$  then
15:        $LC(o_i) \leftarrow LC(o_i) \cup \{o_j\}$ ;
16:        $UC(o_j) \leftarrow UC(o_j) \cup \{o_i\}$ ;
17:     end if
18:   end for
19: end for
20:  $H \leftarrow H \setminus \{o_{old}\}$ ;

```

Fig. 14. *DeleteObject*: removes expired objects from the Hasse diagram

```

Input:  $o_{new}, o_{old}$ 
1: for each arriving object  $o_{new} \in O_i$  do
2:    $InsertObject(o_{new}, H_i)$ ;
3:   Compute a set of combinations that contain  $o_{new}$  and dominate the last combination in ResultSet, NewCandidates with  $|NewCandidates| \leq K$ ;
4:   for each combination  $c \in NewCandidates$  do
5:     Insert  $c$  to ResultSet such that the combinations in ResultSet are in descending order of merged quality assessments;
6:   end for
7: end for
8: for each expired object  $o_{old} \in O_i$  do
9:    $DeleteObject(o_{old}, H_i)$ ;
10:  for each combination  $c \in ResultSet$  that consist of  $o_{old}$  do
11:     $ResultSet \leftarrow ResultSet \setminus \{c\}$ ;
12:  end for
13: end for
14: if  $|ResultSet| < K$  then
15:   Invoke the computation procedure presented in Section V;
16: end if

```

Fig. 15. Incremental maintenance of the *ResultSet*

and therefore need to be eliminated from this set,

- some combinations expire because the objects in them violate windowing constraints, and
- some combinations are dropped because newly arriving objects provide better combinations.

When an object o_{old} expires, it is removed from the Hasse diagram; the combinations in *ResultSet* that contain o_{old} are also eliminated.

Upon arrival of a new object, o_{new} , to queue in_qu_i , new combinations with o_{new} are progressively computed until a combination dominated by the last combination in the previous *ResultSet* (before the expiration of objects and combinations) is found: We first identify new candidate results, *NewCandidates*: these are the m -inputs, formed by combining o_{new} within $maxset_j, j \neq i$, that dominate the last combination in the previous *ResultSet*. At this point, the combinations in *NewCandidates* are as good or better than the combinations currently in the *ResultSet*. If the cardinality of *NewCandidates* is greater than K , only the first K such combinations are enumerated. The *ResultSet* is then merged with the *NewCandidates* set in descending order of merged quality assessments. Note that the cardinality of *ResultSet* can be greater than K after it is merged with *NewCandidates*. In this case, only the first K combinations are considered.

If after the above steps, the number of objects in the *ResultSet* is still below K , the algorithm in Section V-A is invoked to fill the rest of the result set.

E. Complexity of the Incremental Algorithm

The complexity of the incremental algorithm depends on the number of objects expired in each window and how many of the K candidates in the *ResultSet* have been consumed by the fusion operator. In the worst case, no objects have been expired and all combinations in the *ResultSet* have been consumed (thus the entire set needs to be replenished).

The worst case time complexity of inserting a new object into an existing Hasse diagram for the i^{th} stream is $O(|O_i|)$: the new object has to be compared with all existing objects at least once. The space complexity of the process is also $O(|O_i|)$ in that the sets UC and LC can together contain the objects in the entire window.

Once the incremental maintenance of the Hasse diagram is over, the maintenance of the *ResultSet* starts. In the worst case, the *ResultSet* is empty and the new object does not result in any single promising combination (i.e., better than the worst one just expired). In this case, the basic algorithm has to be executed from scratch (over the incrementally updated Hasse diagrams) to identify K new combinations to be passed to the fusion operator.

VI. EXPERIMENTS

In this section, we evaluate the performance of the combination shedding algorithms (both incremental and non-incremental) and the quality of service that they provide. More specifically, we compare the performance of the incremental combination shedding scheme presented in this paper with the performance of several other approaches, including non-incremental combination shedding (which, for each window shift, recreates the Hasse diagram and re-computes the best K combinations from scratch), an *object-based* elimination scheme (OBJ-Fair), where the $\lceil n^{\frac{1}{K}} \rceil$ best objects from each input stream are fused and the rest are shed, an object-based random elimination scheme (OBJ-Random), where $\lceil n^{\frac{1}{K}} \rceil$ objects are randomly selected from each input stream are fused and the rest are shed, an object-based *greedy* elimination scheme (OBJ-Greedy), where we select the best objects in all streams in such a way that the selected objects will lead to K combinations. For comparing fusion qualities of the different schemes with the best possible fusion qualities that can be obtained, we also consider an exhaustive combination enumeration (ENUM) algorithm which selects the best K combinations within the current window.

A. Setup

For the evaluation of the different schemes, we considered streaming scenarios where input object quality assessments were characterized using normal distributions, each described by a mean and a variance (Example 4.5).

Given two quality assessments, $qa(o_i) = N(q_i, \xi_i)$ and $qa(o_j) = N(q_j, \xi_j)$, $o_i \preceq_p o_j$ iff $q_j \geq q_i \wedge \xi_j \leq \xi_i$. We considered two different quality assessment fusion semantics: *average-based* and *minimum-based*. The average-based (μ_{avg}) quality assessment for a set O , of objects is

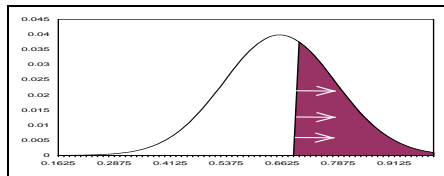


Fig. 16. Confidence score: given a quality assessment, $qa_o = N(q_o, \xi_o)$ and a threshold τ , the confidence value, $\int_{\tau}^{\infty} N(q_o, \xi_o)(q) dq$, is equal to the area under the normal curve beyond τ (in the above example, $\tau = 0.7$)

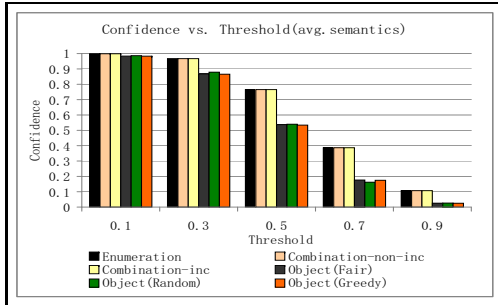


Fig. 17. Quality confidence associated with the fusion results obtained under average fusion semantics for different τ lower-bounds

$N\left(\frac{\sum_{N(q_i, \xi_i) \in Q_O} q_i}{|Q_O|}, \frac{\sum_{N(q_i, \xi_i) \in Q_O} \xi_i}{|Q_O|^2}\right)$, where Q_O denotes the set of quality assessments of the objects in O . The minimum-based (μ_{min}) quality assessment for a set O , of objects is $N(\min\{q_i \mid N(q_i, \xi_i) \in Q_O\}, \max\{\xi_i \mid N(q_i, \xi_i) \in Q_O\})$. The number of input streams to the fusion operators varied between 2 and 4. The means of the quality assessments for the four input streams varied within $[0.5, 0.8]$, $[0.2, 0.6]$, $[0.4, 0.7]$, and $[0.3, 0.6]$, respectively. Their standard deviations were within the range, $[0.1, 0.2]$. Note that this corresponds to a situation where the input media streams are of relatively low quality (the average of the means of the quality assessments is only 0.35). Thus, it is most critical to be able to choose the best combinations for processing.

We varied K between 1 and 30. The window size was up to 100 objects per stream and the degree of ambiguity, f , varied between 3 and 9 incomparable objects per window². We considered three different per-stream window shifts values: 10, 20, or 30 objects per shift. The default settings for the parameters are as follows: the number of streams (m) = 4; the window shift (wsh) = 10; fusion capacity (K) = 20; and window size (W)=100. The experiments are performed on a Pentium PC with 2.80 GHZ CPU and 1.00 GB of RAM.

B. Evaluation Measures

In this experiments, we report the execution times of the various shedding algorithms as well as the qualities of the resulting fused objects. Note that, as described above, the quality assessments of the individual fused objects are normally distributed. Therefore, we compare the quality assessments of the fusion results based on the quality “confidences” they provide: given a fused object o , its quality assessment $qa_o = N(q_o, \xi_o)$, and a threshold, τ , the corresponding *quality confidence* is defined as $\int_{\tau}^{\infty} qa_o(q) dq = \int_{\tau}^{\infty} N(q_o, \xi_o)(q) dq$ (Figure 16). The higher the confidence value is, the more likely it is that the resulting fused object is of high quality.

²Degree of ambiguity refers to the number of incomparable objects in a given stream join window.

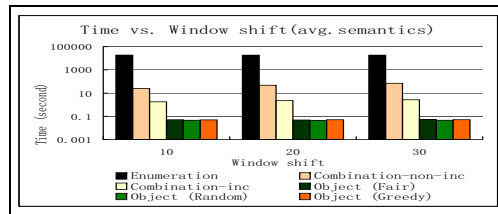


Fig. 18. Performance of various shedding algorithms under average fusion semantics as a function of the window shift

C. Result Qualities

The confidence measure described above is defined with respect to a lower-bound, τ . Before we evaluate the behavior of the shedding schemes under varying system parameters, we first report the effect of τ on the output confidence. Figure 17 shows the confidence values obtained under the default configuration for different τ thresholds. As can be seen here, the result confidences are highly sensitive to τ . In particular, when the lower-bound is set very low, all schemes are able to achieve very high confidence values. When the lower-bound is set higher than the average quality of the incoming objects (~ 0.3), however, the confidence level for all schemes drops as expected. Nevertheless, the quality-based combination shedding schemes presented in this paper are always able to achieve confidence levels as high as the exhaustive enumeration, whereas the object-based shedding schemes fail to match the same level of result qualities. Moreover, for all practical purposes, there is no difference between a random object shedding scheme and quality-aware object shedding scheme. This shows that quality-based combination shedding is especially useful when one aims to maximize the result qualities over incoming data with low quality observations. Even when we set τ to 0.3, which is actually slightly lower than the average object quality of the input streams, the combination shedding schemes provide almost 100% confidence and work significantly better than the object-shedding schemes. In the rest of the paper, we will use this value as the lower-bound for confidence calculations.

D. Effects of the Various System Parameters

As expected, in terms of execution time, the object shedding algorithms are at least an order faster than the combination shedding schemes (Figure 18). As shown before, however, these time savings come with significant losses in quality, and thus the low qualities associated with the results render object-shedding schemes undesirable in multimedia applications. Combination shedding schemes are 3 to 5 orders faster than the full enumeration scheme (i.e., they are able to achieve significant savings in time without giving up result qualities), however the non-incremental algorithm is still too slow for many applications and does not scale well with the size of the window shift. The incremental approach, however, is an order faster than the non-incremental scheme and scales better with the size of the window shift.

As shown in Figures 19 through 21, similar patterns are observed when other parameters are varied as well: the incremental combination shedding scheme is at least an order faster than its non-incremental version. As expected, the object-shedding schemes are much faster, yet their result qualities

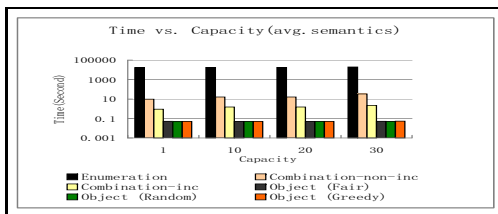


Fig. 19. Performance of various shedding algorithms under average fusion semantics as a function of the fusion capacity (K)

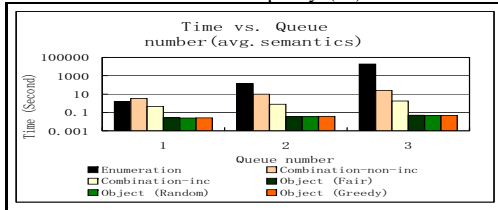


Fig. 20. Performance of various shedding algorithms under average fusion semantics as a function of the number of streams

are always significantly lower than the combination shedding schemes. The execution times of the combination shedding algorithms are most affected by the number of streams and the window size.

E. Effects of the Different Fusion Semantics

Figure 22 shows the output qualities under different fusion semantics (the execution times are not affected by the different fusion semantics, so they are not presented in this section): (a) the average semantics returns the average of the qualities of the selected objects while (b) the minimum semantics uses the quality of the worst object as the quality of a given combination. As can be seen here, since it is more conservative, the minimum semantics results in lower confidence values. However, under both fusion semantics, the combination shedding schemes are able to match the result qualities of those of the full enumeration scheme, whereas the object shedding schemes are always significantly poor in terms of the qualities. The difference is more pronounced (~ 0.7 confidence vs. ~ 0.5 confidence) under the more conservative minimum fusion semantics.

F. Effectiveness in Matching the Fusion Speed

As we have discussed in Section III-B, for a shedding scheme to be effective, its speed has to match the combination processing (i.e., fusion) speed. In particular, given m input streams and window size W , in the steady state, the time, $T(W, K, m)$ to identify K combinations for fusion should be such that $T_{fusion} \geq T(W, K, m)/K$, where T_{fusion} is the time that it takes for the fusion operator to process an individual fusion candidate. In Figures 23(a-d), we are plotting the value, $\frac{T(W, K, m)}{K}$, which (based on the above inequality) represents the fastest fusion operation can be supported by the various shedding schemes. As can be seen here, by increasing K , it is possible to match the processing speeds of faster fusion operators. Number of streams and the window size have the biggest impact on the speed of the combination shedding algorithms. For example, while the incremental combination shedding requires on the order of 100ms per fusion operation ($T_{fusion} \geq T(W, K, m)/K \sim 0.1s$) when the window size is hundred, it is able to match speeds on the order of 10s of milliseconds when the window size is set to 20.

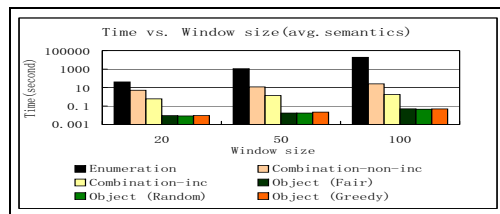
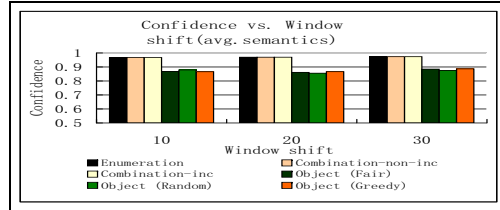
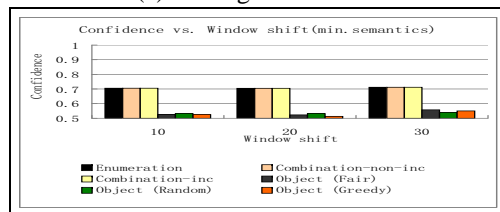


Fig. 21. Performance of various shedding algorithms under average fusion semantics as a function of the window size



(a) Average Semantics



(b) Minimum Semantics

Fig. 22. Output quality performance of various shedding algorithms under different fusion semantics (as a function of the window shift)

G. Effectiveness in Matching the Arrival Rate

As discussed in Section III-B, shedding schemes also have to match the input arrival speeds; i.e., in the steady state, the inequality $T_o \geq T(W, K, m)/wsh$, where T_o denotes the input object inter-arrival time, should hold.

In Figures 23(e-h), we are plotting the value, $\frac{T(W, K, m)}{wsh}$, which represents the fastest input arrival rates that can be supported by the various shedding schemes. By increasing wsh , it is possible to match faster arrival rates. After awhile though, the benefits of window shifting reduces for incremental combination shedding: this is because, if the window shift are large, then more work has to be done from scratch at each iteration. Once again, the number of streams and the window size have impact on the speed of the combination shedding algorithms, but the algorithms presented in this paper are more protected against these variations than the full enumeration.

H. Summary of Experiment Results

Experiments reported in this section show that

- unlike object-based schemes which result in poor overall qualities, combination shedding schemes achieve confidence levels as high as the exhaustive enumeration;
- factors that have the greatest impact on the cost of combination shedding are the number of streams and the window size; yet combination shedding is multiple orders faster than exhaustive enumeration and, moreover, incremental combination shedding is an order faster than the non-incremental version;
- by varying the parameter K , it is possible to match the processing speeds of fusion operators; and
- by varying wsh , it is possible to match object arrival rates.

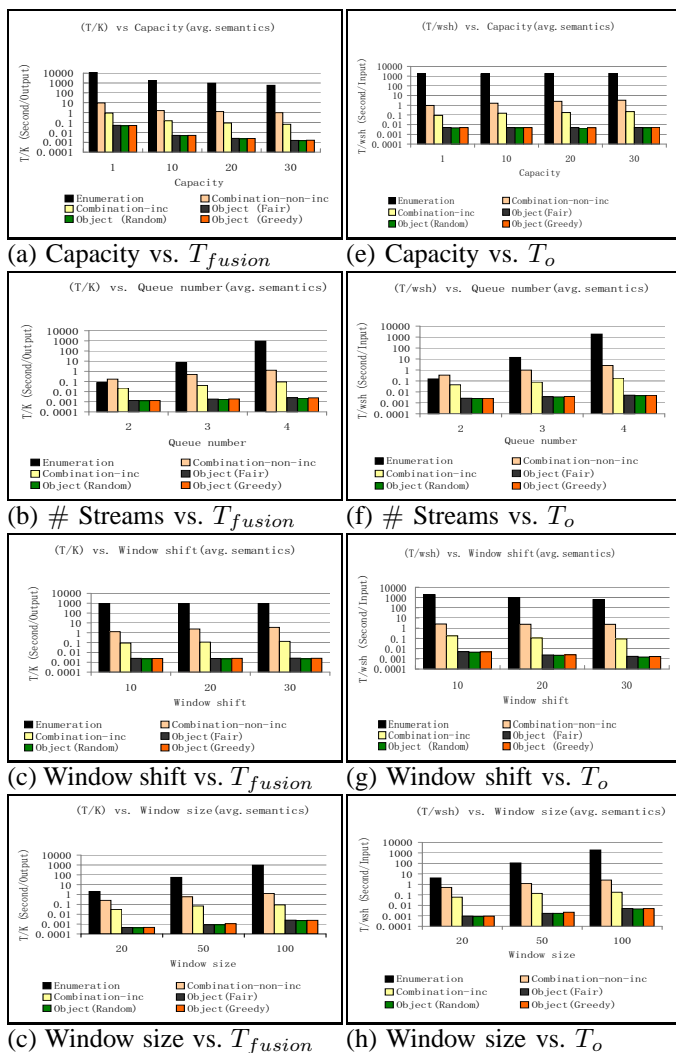


Fig. 23. Effectiveness of various shedding alternatives in terms of matching the processing time (T_{fusion}) and the object inter-arrival time (T_o)

VII. CONCLUSION

In this paper, we highlighted that media fusion operators are expensive, quality assessments of the objects are variable, and the goal of the media processing workflow is not to maximize the number of the output results, but to maximize their qualities. With these in mind, we presented quality-assessment models for objects and fusion operators. We showed that in many cases quality assessments are partially ordered and that unambiguously monotonic ranking functions may not exist. We highlighted that these prevent simple object shedding schemes as well as application of now-standard top- K type ranking algorithms to select the best result combinations. Thus, we developed novel load shedding schemes applicable under these conditions and showed their correctness.

REFERENCES

- [1] M. Akdere, U. Çetintemel, D. Crispell, J. Jannotti, J. Mao, and G. Taubin, "Data-centric visual sensor networks for 3d sensing," in *Intl. Conf. on Geosensor Networks*, 2006.
- [2] B. Liu, A. Gupta, and R. Jain, "Med sman: a streaming data management system over live multimedia," in *ACM Multimedia*, 2005, pp. 171–180.
- [3] K. Nahrstedt and W. T. Balke, "A taxonomy for multimedia service composition," in *ACM MM*, 2004, pp. 88–95.

- [4] C. Intanagonwivat, D. Estrin, R. Govindan, and J. S. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *ICDCS*, 2002, p. 457.
- [5] Q. Liang, "Power aware video traffic classification in the compression domain," in *IEEE Mil. Com. Conf.*, vol. 2, 2002, pp. 1160 – 1164.
- [6] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, 1999, p. 1150.
- [7] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, 2005.
- [8] A. Das, J. Gehrke, and M. Riedewald, "Approximate join processing over data streams," in *ACM SIGMOD*, 2003, pp. 40 – 51.
- [9] N. Tatbul, U. Çetintemel, M. Cherniack, M. Stonebraker, and S. Zdonik, "Load shedding in a data stream manager," in *VLDB*, 2003, pp. 309 – 320.
- [10] J. Xie, J. Yang, and Y. Chen, "On joining and caching stochastic streams," in *ACM SIGMOD*, 2005, pp. 359–370.
- [11] U. Srivastava and J. Widom, "Memory-limited execution of windowed stream joins," in *VLDB*, 2004, pp. 324–335.
- [12] S. Babu, M. Datar, and R. Motwani, "Load shedding for aggregation queries over data streams," in *ICDE*, 2004, p. 350.
- [13] N. Tatbul and S. Zdonik, "Window-aware load shedding for aggregation queries over data streams," in *VLDB*, 2006.
- [14] Y. Xing, J. Hwang, U. Çetintemel, and S. Zdonik, "Providing resiliency to load variations in distributed stream processing," in *VLDB*, 2006.
- [15] X. Gu and P. Yu, "Adaptive load diffusion for stream joins," in *Middleware*, 2005.
- [16] Z. Abrams and J. Liu, "Greedy is good: On service tree placement for in-network stream processing," in *ICDCS*, 2006, p. 72.
- [17] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure, "Adaptive control of extreme-scale stream processing systems," in *ICDCS*, 2006, p. 71.
- [18] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.
- [19] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *ACM SIGMOD*, 2003, pp. 551–562.
- [20] R. Fagin, "Fuzzy queries in multimedia database systems," in *PODS*, 1998.
- [21] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001, pp. 102–113.
- [22] N. Mamoulis, K. H. Cheng, M. L. Yiu, and D. W. Cheung, "Efficient aggregation of ranked inputs," in *ICDE*, 2006.
- [23] D. Xin, J. Han, and K. C. Chang, "Progressive and selective merge: Computing top-k with ad-hoc ranking functions," in *ACM SIGMOD*, 2007, pp. 775 – 786.
- [24] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *ACM SIGMOD*, 2003, pp. 467–478.
- [25] M. L. Yiu and N. Mamoulis, "Efficient processing of top-k dominating queries on multi-dimensional data," in *VLDB*, 2007, pp. 483–494.
- [26] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," in *SIGMOD*, 2006, pp. 635 – 646.
- [27] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows," in *IEEE ICDE*, 2005, pp. 502 – 513.
- [28] I. Bartolini, P. Ciaccia, V. Oria, and M. T. Ozsu, "Integrating the results of multimedia sub-queries using qualitative preferences," in *Workshop of Multimedia Information Systems*, 2004.
- [29] C. Y. Chan, P. K. Eng, and K. L. Tan, "Stratified computation of skylines with partially-ordered domains," in *ACM SIGMOD*, 2005, pp. 203–214.
- [30] R. Avnur and J. Hellerstein, "Eddies: Continuously adaptive query processing," in *SIGMOD*, 2000, pp. 261–272.
- [31] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman, "Continuously adaptive continuous queries over streams," in *SIGMOD*, 2002, pp. 49–60.
- [32] M. Hammad, W. Aref, and A. Elmagarmid, "Stream window join: Tracking moving objects in sensor-network databases," in *SSDBM*, 2003, pp. 75–84.
- [33] L. Zadeh, "Fuzzy sets," *Information and Control*, pp. 338–H353, 1965.
- [34] —, "The concept of a linguistic variable and its application to approximate reasoning - i," *Information Sciences*, vol. 8, pp. 199–249, 1975.
- [35] S. L. Dockstader and A. M. Tekalp, "Multiple camera fusion for multi-object tracking," in *IEEE Workshop on Multi-Object Tracking*, 2001, p. 95.

APPENDIX

Theorem 5.2 (Complexity): Given the sets, O_1, \dots, O_m , of objects, let f denote the largest mutually-incomparable subset of objects any O_i contains. Let also $n = \max\{|O_1|, \dots, |O_m|\}$. Let t_{eval} be the time needed for evaluating the merged quality assessment of a given m -tuple. Then, the algorithm runs in $O(mn^2 + mf^m \log f + K^2 m^2 f + (K^2 + mfK) \log(K + mf) + (mfK + f^m)t_{eval})$.

The algorithm uses $O(Km + m^2 f + f(\sum_{1 \leq i \leq m} |O_i|))$ runtime space.

Proof: The cost of creating a partial order graph for each input stream of length n is at most $O(n^2)$. Thus, for m streams, this requires $O(mn^2)$ time.

To construct $C[1]$, we first take the Cartesian product of the roots of the input partial order graphs. If each graph can have at most f roots, the cost of this step is $O(f^m)$.

Then, the cost of selecting the best ranking K among the f^m candidates in $C[1]$ requires (if sorting used) $O(mf^m \log f + f^m t_{eval})$ time. If K is small, it is also possible to implement this step at $O(Kf^m + f^m t_{eval})$ time.

Each iteration identifies at most $O(mf)$ new candidates. Each new candidate needs to be checked for redundancy against the already existing $O(K)$ candidates (Step 5.c.iii). Each check costs $O(m)$ time: a total of m quality assessment comparisons are required, one for each object in the candidate m -tuple. Thus, the redundancy check of $O(mf)$ new candidates against $O(K)$ candidates has a running time of $O(m^2 f K)$. At the end of the iteration, keeping at most K m -tuples with highest quality assessments require evaluating the merged assessments of the new candidates (in $O(mf t_{eval})$ time) and sorting all the candidates in $O((K + mf) \log(K + mf))$ time. K iterations, then, lead to a running time of $O(K(m^2 f K + m f t_{eval} + (K + mf) \log(K + mf)))$.

In terms of space utilization, the algorithm has to create m Hasse diagrams each of which will hold one node per object in the current window and at most f edges per node. Therefore, the total size of the Hasse diagrams is bounded by $f(\sum_{1 \leq i \leq m} |O_i|)$. In addition, the algorithm needs to hold $K + mf$ candidates (m -tuples) in memory during each iteration. Since each combination is of m length, the space requirement for candidate combinations is $O(Km + m^2 f)$. ■