9-2009

Analysis of Uncertain Data: Smoothing of Histograms

Eugene Fink
Carnegie Mellon University, e.fink@cs.cmu.edu

Ankur Sarin
Carnegie Mellon University

Jaime G. Carbonell
Carnegie Mellon University, jgc@cs.cmu.edu

Published In
Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2549- 2555.

Analysis of Uncertain Data: Smoothing of Histograms

Eugene Fink
e.fink@cs.cmu.edu
www.cs.cmu.edu/~eugene

Ankur Sarin
asarin@andrew.cmu.edu

Jaime G. Carbonell
jgc@cs.cmu.edu
www.cs.cmu.edu/~jgc

Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, United States

***Abstract*—We consider the problem of converting a set of numeric data points into a smoothed approximation of the underlying probability distribution. We describe a representation of distributions by histograms with variable-width bars, and give a greedy smoothing algorithm based on this representation.**

***Keywords*—Probability distributions, smoothing, compression, visualization, histograms.**

I. Introduction

When analyzing a collection of numeric measurements, we often need to represent it as a histogram that shows the underlying distribution. For example, consider the set of 5000 numeric points in Figure 9(a) on the next-to-last page of the paper. We may convert it to a fine-grained histogram in Figure 9(b), to a coarser version in Figure 9(c), or to a very coarse histogram in Figure 9(d). This conversion helps to visualize data, and it also serves as lossy data compression, since it allows replacing a large point set with a compact approximation of the underlying distribution.

We describe a mechanism for constructing an approximate probability density function based on a given point set, which represents this function by a generalized histogram with variable-width bars. It also supports lossy compression of a fine-grained histogram into a coarser version. We formalize the problem of smoothing a histogram (Section II), give algorithms for solving it (Sections III–V), discuss their extensions (Section VI), and present initial empirical results (Section VII).

This work has been part of a project aimed at development of techniques for automated reasoning under uncertainty, including representation of insufficient and approximate data [Bardak et al., 2006a; Fu et al., 2008]; optimization and hypothesis evaluation based on limited data [Fink et al., 2006; Gershman et al., 2009a]; and planning of additional data gathering [Bardak et al., 2006b; Gershman et al., 2009b].

The manuscript was received on March 31, 2009. The described work was supported by the Air Force Research Laboratory (AFRL) under Contract No. FA8750-07-2-0137.

II. Problem

We consider the task of smoothing a probability distribution. Specifically, we need to develop a procedure that inputs a point set or a fine-grained histogram, and constructs a coarser-grained histogram. We explain the representation of probability distributions in the developed system and then formalize the smoothing problem.

Representation: We encode an approximated probability density function by a histogram with variable-width bars, which is called a *piecewise-uniform distribution* and abbreviated as *PU-distribution*. This representation is a set of disjoint intervals, with a probability assigned to each interval, as shown in Figure 1 [Bardak et al., 2006a; Fu et al., 2008]. We assume that all values *within* a specific interval are equally likely; thus, every interval is a uniform distribution, and the histogram comprises multiple such distributions.

Note that a standard histogram is a special case of a PU-distribution, where all intervals have the same width and the distances between adjacent intervals are zero. Furthermore, a set of numeric points is also a special case of such a distribution, where all intervals have zero width.

Inputs: The smoothing procedure accepts a PU-distribution and generates a version with fewer intervals, which should be "close" to the initial distribution according to a given distance function. The inputs of this procedure are as follows.

- ***Initial distribution:*** We input a PU-distribution with m intervals, which may represent a point set or a generalized histogram, encoded by three real-valued arrays:

initial-prob[1..m]	interval probabilities
initial-min[1..m]	left endpoints of intervals
initial-max[1..m]	right endpoints of intervals

- ***Distance function:*** We provide a two-argument function, denoted *dist-func*, which determines distances between distributions. It inputs two PU-distributions and returns a numeric distance between them. We may use the Kullback-Leibler divergence [Kullback and Leibler, 1951; Kullback, 1987], the Jensen-Shannon divergence [Lin, 1991], or any other distance measure.
- ***Target size:*** We specify the required number of intervals in the smoothed distribution, denoted k, which must be smaller than the size m of the initial distribution.

Problem: The goal is to produce a PU-distribution, with the given number of intervals, that is as close as possible to the initial distribution according to the given distance function.

III. Greedy algorithm

We now describe a smoothing procedure; we summarize its main data structures in Figure 2 and give its pseudocode in Figure 3. It gradually reduces the number of intervals in a PU-distribution by merging adjacent intervals. At each step, it considers all possible merges and selects the merge that leads to the smallest "change," as measured by the distance function. It stops upon reaching the target number of intervals.

This algorithm is greedy, and it does *not* guarantee finding the k-interval PU-distribution that is globally closest to the initial distribution; however, it gives good results in practice and usually constructs a near-optimal smoothing.

The procedure performs $(m - k)$ merging steps, and considers $O(m)$ candidate merges at each step (see the SMOOTH subroutine in Figure 3). For each candidate merge, it computes the distance between the current and the after-merge distributions (see the MERGE-DIST subroutine in Figure 3), which takes $O(m)$ time for standard distance functions. Thus, a straightforward implementation of the smoothing procedure would have $O(m^3)$ running time. Fortunately, most distance functions have a "local-computation" property, discussed in Section IV, which allows the reduction of the running time to $O(m \cdot \log m)$.

IV. Faster algorithm

The main inefficiency of the algorithm in Figure 3 comes from the MERGE-DIST subroutine, which takes $O(m)$ time to evaluate a potential merge by computing the distance between the current and the after-merge distributions. For most distance measures, we may perform this computation "locally," by looking only at the two merged intervals, rather than applying the distance function to the entire PU-distribution.

For example, consider a simple measure of the distance between two probability density functions, $p(x)$ and $q(x)$, defined as the mean absolute difference:

$$\textit{simple-dist}(p, q) = \int |\, p(x) - q(x) \,|\, dx.$$

If we use it, we can compute the distance between the current and the after-merge PU-distributions locally, using the SIMPLE-MERGE-DIST subroutine in Figure 4. Its running time is constant, which reduced the overall time of the smoothing algorithm to $O(m^2)$.

We can further improve the efficiency by arranging the potential merges in a priority queue. Specifically, we define the *merge distance* of an interval, denoted *m-dist*[*bar*] in the pseudocode, which is the distance between the current PU-distribution and the distribution that would result from merging the given interval, *bar*, with the next one, *next*[*bar*]. We compute these distances for all intervals except the last one, and put them in a priority queue. At each step, we extract the smallest-distance interval from the queue and merge it with the next interval. We give this fast smoothing procedure in Figure 5; its running time is $O(m \cdot \log m)$.

Note that the only logarithmic-time operations within the main loop of the fast smoothing procedure are the priority-queue operations. The other computations within the main loop take constant time. In practice, however, the priority-queue operations are very fast even when the queue size is in hundreds of millions, and they take less than 10% of the overall time. Thus, the "practical" running time of the computations within the main loop is near-constant, which means that the "practical" time of the overall smoothing procedure is near-linear, that is, close to O(m).

V. Alternative distance measures

We review two other standard distance functions that allow fast computation of merge distances. For both functions, the theoretical complexity of the smoothing procedure is $O(m \cdot \log m)$, but the practical smoothing time is near-linear.

Kullback–Leibler: This distance function is a standard measure of divergence between two distributions, also called *information gain* or *relative entropy*:

$$\textit{KL-dist}(p, q) = \int p(x) \cdot \log \big(p(x) \,/\, q(x)\big)\, dx.$$

In Figure 4, we give a constant-time subroutine, called KL-MERGE-DIST, for computing the related merge distances.

Jensen–Shannon: It is another standard divergence measure, also called *information radius*, which is defined through the Kullback-Leibler divergence:

$$\textit{JS-dist}(p, q) = \big(\textit{KL-dist}(p, (p+q)/2) + \textit{KL-dist}(q, (p+q)/2)\big) \,/\, 2.$$

In Figure 4, we show the related distance-computation subroutine, called JS-MERGE-DIST.

VI. Extensions

We discuss extensions to the smoothing technique, which allow adapting it to a wider range of practical problems.

Termination conditions: We have described an algorithm that stops only upon reaching the target number of intervals, but we have included two additional termination conditions in the implemented system. The user may optionally specify an upper bound on the distance between the initial and the smoothed distribution, and she may also set an upper bound on a single-step merge distance. When the system reaches either bound, it stops even if the PU-distribution has more than the target number of intervals. These bounds allow the user to limit the accuracy loss due to the smoothing.

Spline fitting: When presenting a PU-distribution to the user, we may display a smooth curve rather than a histogram [Parzen, 1962; Boneva et al., 1971], which is a better visual aid. We generate it by fitting a spline over the PU-distribution in such a way that, for each interval, the area under the spline is equal to the interval's probability (Figure 6). We use standard spline tools; for example, see the "smoothing a histogram" demo for Spline Toolbox 3.3.6 at www.mathworks.com/products/splines/demos.html.

Piecewise-uniform distribution

We describe a probability density function by multiple intervals, and we specify each interval by its probability along with its minimal and maximal values:

$prob_1$: from min_1 to max_1
$prob_2$: from min_2 to max_2
…
$prob_m$: from min_m to max_m

The intervals do not overlap, and the sum of their probabilities is 1.0, which means that we impose the following constraints:

$min_1 \le max_1 \le min_2 \le max_2 \le \ldots \le min_m \le max_m$;
$prob_1 + prob_2 + \ldots + prob_m = 1.0$.

Important special cases

- ***Standard histogram:***
  $max_1 - min_1 = max_2 - min_2 = \ldots = max_m - min_m$;
  $max_1 = min_2, max_2 = min_3, \ldots, max_{m-1} = min_m$.
- ***Set of numeric data points:***
  $min_1 = max_1, min_2 = max_2, \ldots, min_m = max_m$.

Figure 1: Encoding of a *PU-distribution*, which is a generalized histogram with variable-width bars.

Interval

We represent an *interval* (also called a *bar*) of a PU-distribution by a structure that includes its probability, left and right endpoints, pointers to the previous and next intervals, and the related *merge distance*, defined in Section IV. We use the word *bar* to name the variables of this type in the pseudocode, and refer to the following fields of such variables:

prob[*bar*]	probability of the interval
min[*bar*]	left endpoint
max[*bar*]	right endpoint
prev[*bar*]	previous interval in the PU-distribution
next[*bar*]	next interval in the PU-distribution
m-dist[*bar*]	*merge distance*, defined in Section IV

PU-distribution

We represent a *PU-distribution* (also called a *histogram*) by a structure that includes a doubly linked list of intervals in the sorted order. We use the word *hist* to name the variables of this type, and refer to the following fields of such variables:

size[*hist*]	number of intervals in the list
first[*hist*]	pointer to the first interval
last[*hist*]	pointer to the last interval

Figure 2: Main data structures in the smoothing procedure.

The subroutine inputs three arrays that represent an initial distribution (see Section II) and constructs the respective PU-distribution structure, which is a doubly linked list of *m* interval structures.

```
INITIAL-HIST(initial-prob, initial-min, initial-max, m)
hist = new PU-distribution
first[hist] = new interval; prob[first[hist]] = initial-prob[1]
min[first[hist]] = initial-min[1]; max[first[hist]] = initial-max[1]
prev[first[hist]] = NIL; last[hist] = first[hist]
for i = 2 to m do
    bar = new interval; prob[bar] = initial-prob[i]
    min[bar] = initial-min[i]; max[bar] = initial-max[i]
    prev[bar] = last[hist]; next[last[hist]] = bar; last[hist] = bar
next[last[hist]] = NIL
size[hist] = m
return hist
```

The subroutine inputs an interval, *bar*. It merges this interval with the next, and returns the resulting merged interval.

```
MERGED-BARS(bar)
new-bar = new interval
prob[new-bar] = prob[bar] + prob[next[bar]]
min[new-bar] = min[bar]; max[new-bar] = max[next[bar]]
prev[new-bar] = prev[bar]; next[new-bar] = next[next[bar]]
return new-bar
```

The subroutine inputs a PU-distribution, *hist*, and a pointer to one of its intervals, *bar*. It merges this interval with the next, and replaces the two original intervals with the merged interval.

```
MERGING(hist, bar)
new-bar = MERGED-BARS(bar)
if first[hist] = bar
    then first[hist] = new-bar; else next[prev[bar]] = new-bar
if last[hist] = next[bar]
    then last[hist] = new-bar; else prev[next[next[bar]]] = new-bar
delete next[bar]; delete bar
size[hist] = size[hist] − 1
```

The subroutine inputs a PU-distribution, *hist*; a pointer to one of its intervals, *bar*; and a distance function, *dist-func*. It computes the distance between *hist* and the new distribution that would result from merging *bar* with the next interval.

```
MERGE-DIST(hist, bar, dist-func)
create a new PU-distribution, copy-hist, which is a copy of hist;
    this operation includes copying all intervals of hist
let copy-bar be the copy of bar in the new distribution
MERGING(copy-hist, copy-bar)
dist = dist-func(hist, copy-hist)
delete the PU-distribution copy-hist and all its intervals
return dist
```

The procedure inputs an initial PU-distribution, represented by the arrays *initial-prob*, *initial-min*, and *initial-max*, along with their size *m*; a distance function, *dist-func*; and a target distribution size *k*. It generates a smoothed distribution with *k* intervals.

```
SMOOTH(initial-prob, initial-min, initial-max, m, dist-func, k)
hist = INITIAL-HIST(initial-prob, initial-min, initial-max, m)
while size[init] > k do
    bar = first[hist]; best-dist = $+\infty$
    while next[bar] $\ne$ NIL do
        dist = MERGE-DIST(hist, bar, dist-func)
        if dist < best-dist
            then best-bar = bar; best-dist = dist
        bar = next[bar]
    MERGING(hist, best-bar)
return hist
```

Figure 3: Smoothing of a PU-distribution. The procedure inputs an initial distribution and produces its smoothed version.

Auxiliary subroutine for calculating the *height* of an interval, which corresponds to the visual height of the respective histogram bar.

```
HEIGHT($bar$)
return $prob[bar] / (max[bar] - min[bar])$
```

Computation of the simple distance, which is the mean absolute difference between probability density functions (Section IV).

```
SIMPLE-MERGE-DIST($bar$)
$new\text{-}bar$ = MERGED-BARS($bar$)
$dist = |\text{HEIGHT}(new\text{-}bar) - \text{HEIGHT}(bar)| \cdot (max[bar] - min[bar])$
        $+\ \text{HEIGHT}(new\text{-}bar) \cdot (min[next[bar]] - max[bar])$
        $+\ |\text{HEIGHT}(new\text{-}bar) - \text{HEIGHT}(next[bar])|$
          $\cdot\ (max[next[bar]] - min[next[bar]])$
delete $new\text{-}bar$
return $dist$
```

Computation of the Kullback-Leibler divergence (Section V).

```
KL-MERGE-DIST($bar$)
$new\text{-}bar$ = MERGED-BARS($bar$)
$dist = \text{HEIGHT}(bar)$
          $\cdot\ \log\big(\text{HEIGHT}(bar) / \text{HEIGHT}(new\text{-}bar)\big)$
          $\cdot\ (max[bar] - min[bar])$
        $+\ \text{HEIGHT}(next[bar])$
          $\cdot\ \log\big(\text{HEIGHT}(next[bar]) / \text{HEIGHT}(new\text{-}bar)\big)$
          $\cdot\ (max[next[bar]] - min[next[bar]])$
delete $new\text{-}bar$
return $dist$
```

Computation of the Jensen-Shannon divergence (Section V).

```
JS-MERGE-DIST($bar$)
$new\text{-}bar$ = MERGED-BARS($bar$)
$left\text{-}mean\text{-}height = \big(\text{HEIGHT}(bar) + \text{HEIGHT}(new\text{-}bar)\big) / 2$
$right\text{-}mean\text{-}height = \big(\text{HEIGHT}(next[bar]) + \text{HEIGHT}(new\text{-}bar)\big) / 2$
$kl\text{-}dist_1 = \text{HEIGHT}(bar)$
          $\cdot\ \log\big(\text{HEIGHT}(bar) / left\text{-}mean\text{-}height\big)$
          $\cdot\ (max[bar] - min[bar])$
        $+\ \text{HEIGHT}(next[bar])$
          $\cdot\ \log\big(\text{HEIGHT}(next[bar]) / right\text{-}mean\text{-}height\big)$
          $\cdot\ (max[next[bar]] - min[next[bar]])$
$kl\text{-}dist_2 = \text{HEIGHT}(new\text{-}bar)$
        $\cdot\ \Big(\log\big(\text{HEIGHT}(new\text{-}bar) / left\text{-}mean\text{-}height\big)$
            $\cdot\ (max[bar] - min[bar])$
          $+\ min[next[bar]] - max[bar]$
          $+\ \log\big(\text{HEIGHT}(new\text{-}bar) / right\text{-}mean\text{-}height\big)$
            $\cdot\ (max[next[bar]] - min[next[bar]])\Big)$
delete $new\text{-}bar$
return $(kl\text{-}dist_1 + kl\text{-}dist_2) / 2$
```

Figure 4: Constant-time distance computation. These subroutines are efficient versions of MERGE-DIST, given in Figure 3, for three specific distance functions. They calculate the distance between the current distribution and the new one that would result from merging two adjacent intervals.

The subroutine inputs an interval, *bar*, and a distance function, *dist-func*. It calls the appropriate code in Figure 4 for the local computation of the merge distance. Note that we have to implement a specialized subroutine for each distance function.

```
FAST-MERGE-DIST($bar$, $dist\text{-}func$)
if $dist\text{-}func$ is the simple distance from Section IV
  then return SIMPLE-MERGE-DIST($bar$)
if $dist\text{-}func$ is the Kullback-Leibler divergence
  then return KL-MERGE-DIST($bar$)
if $dist\text{-}func$ is the Jensen-Shannon divergence
  then return JS-MERGE-DIST($bar$)
```

The subroutine inputs an initial PU-distribution, *hist*, and a distance function, *dist-func*. It creates a priority queue of intervals, prioritized by the merge distance.

```
INITIAL-QUEUE($hist$, $dist\text{-}func$)
create an empty priority queue, denoted $queue$,
  which prioritizes intervals by the merge distance
$bar = first[hist]$
while $next[bar] \neq$ NIL do
  $m\text{-}dist[bar]$ = FAST-MERGE-DIST($bar$, $dist\text{-}func$)
  add $bar$ to $queue$, prioritized by $m\text{-}dist$
  $bar = next[bar]$
return $queue$
```

The subroutine inputs a PU-distribution, *hist*; a pointer to one of its intervals, *bar*; a distance function, *dist-func*; and a priority queue of intervals, *queue*. It merges the given interval with the next, replaces these two intervals with the merged interval, and updates the queue.

```
FAST-MERGING($hist$, $bar$, $dist\text{-}func$, $queue$)
$new\text{-}bar$ = MERGED-BARS($bar$)
remove $bar$ from $queue$
if $first[hist] = bar$
  then $first[hist] = new\text{-}bar$
  else $next[prev[bar]] = new\text{-}bar$
       $m\text{-}dist[prev[bar]]$ = FAST-MERGE-DIST($prev[bar]$, $dist\text{-}func$)
       update position of $prev[bar]$ in $queue$, prioritized by $m\text{-}dist$
if $last[hist] = next[bar]$
  then $last[hist] = new\text{-}bar$
  else $prev[next[next[bar]]] = new\text{-}bar$
       remove $next[bar]$ from $queue$
       $m\text{-}dist[new\text{-}bar]$ = FAST-MERGE-DIST($new\text{-}bar$, $dist\text{-}func$)
       add $new\text{-}bar$ to $queue$, prioritized by $m\text{-}dist$
delete $next[bar]$; delete $bar$
$size[hist] = size[hist] - 1$
```

The procedure inputs an initial PU-distribution, represented by the arrays *initial-prob*, *initial-min*, and *initial-max*, along with their size *m*; a distance function, *dist-func*; and a target distribution size *k*. It generates a smoothed distribution with *k* intervals.

```
FAST-SMOOTH($initial\text{-}prob$, $initial\text{-}min$, $initial\text{-}max$, $m$, $dist\text{-}func$, $k$)
$hist$ = INITIAL-HIST($initial\text{-}prob$, $initial\text{-}min$, $initial\text{-}max$, $m$)
$queue$ = INITIAL-QUEUE($hist$)
while $size[init] > k$ do
  set $best\text{-}bar$ to the lowest-distance interval in $queue$,
    and remove it from $queue$
  FAST-MERGING($hist$, $best\text{-}bar$)
return $hist$
```

Figure 5: Fast smoothing of a PU-distribution. The procedure arranges potential merges in a priority queue. It works only when we can use the local distance computation, described in Section IV.

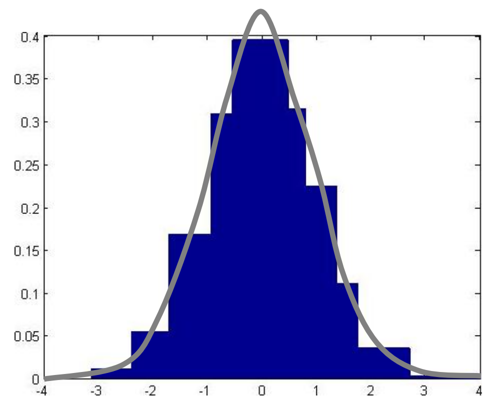


Figure 6: Fitting a spline over a PU-distribution. We use splines to provide a better visualization of the underlying probability density function.

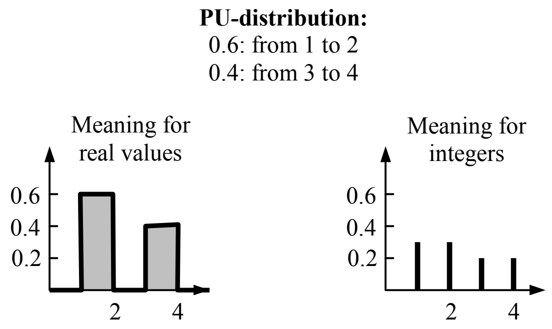


Figure 7: Difference between integer and real-valued PU-distributions. The meaning of a specific PU-encoding depends on whether it describes discrete or continuous values.

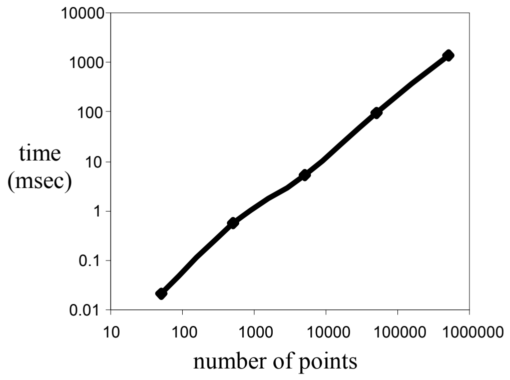


Figure 8: Efficiency of smoothing. We show the dependency of the running time on the number of points in the input set. For m points, the time is $(2.5 \pm 0.5) \cdot 10^{-3} \cdot m$ milliseconds.

Integer PU-distributions: We have described PU-distributions of real values, but we can also use this representation for integers. Note however that the meaning of a specific PU-encoding for integers is *different* from the meaning of the same encoding for reals.

For example, consider the two-interval PU-encoding in Figure 7. If it refers to reals, then it represents a probability density function that consists of two uniform distributions (Figure 7, left). On the other hand, if it is integer, the distribution comprises four discrete values (Figure 7, right).

When using a PU-distribution of integers, we do *not* allow intervals to have common endpoints; thus, the related restriction is somewhat different from that for real values:

- ***Real-valued PU-distributions:***
  $min_1 \le max_1 \le min_2 \le max_2 \le \ldots \le min_m \le max_m$.
- ***Integer PU-distributions:***
  $min_1, max_1, min_2, max_2, \ldots, min_m, max_m$ are integers;
  $min_1 \le max_1 < min_2 \le max_2 < \ldots < min_m \le max_m$.

The distance computations in Figure 4 do not work directly for integer PU-distributions, but we can adapt them using a simple trick. We convert an integer distribution to a real-valued version with similar mathematical properties, which involves extending each interval by 0.5 on both sides:

Given integer distribution:

$prob_1$: from min_1 to max_1
$prob_2$: from min_2 to max_2
…
$prob_m$: from min_m to max_m

Respective real-valued distribution:

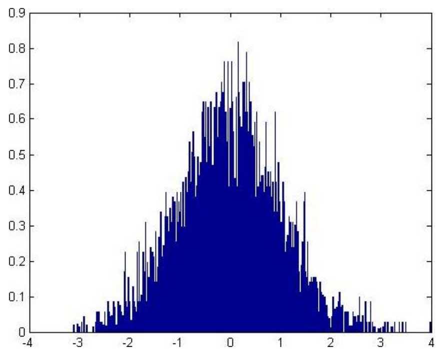
$prob_1$: from $(min_1 - 0.5)$ to $(max_1 + 0.5)$
$prob_2$: from $(min_2 - 0.5)$ to $(max_2 + 0.5)$
…
$prob_m$: from $(min_m - 0.5)$ to $(max_m + 0.5)$

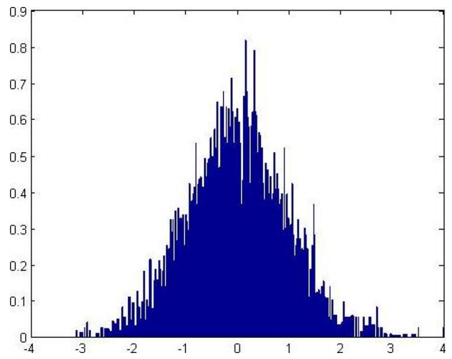
We apply the smoothing procedure to this real-valued version and then convert the output PU-distribution back to integers, which involves trimming all its intervals by 0.5 on each side.

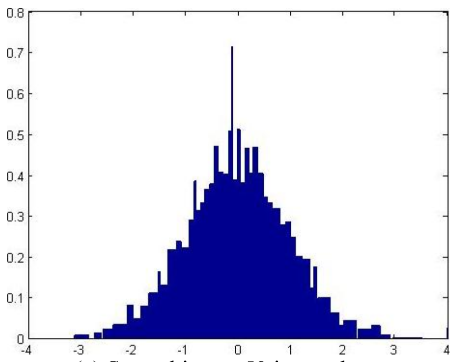
This conversion preserves the correctness of all mathematical computations in Figure 4. Thus, it allows the application of the described algorithms to integer PU-distributions without loss of accuracy.

VII. Initial Experiments

We have applied the smoothing procedure to point sets drawn from several standard distributions, including normal, geometric, Poisson, uniform, and binomial. It has produced reasonable, visually appealing results in all cases; we have not observed any difference in its effectiveness among different distributions. We show examples of smoothing for the normal distribution and the geometric distribution (Figures 9 and 10).


(a) Initial set of 5000 points.


(b) Smoothing to 200 intervals.


(c) Smoothing to 50 intervals.

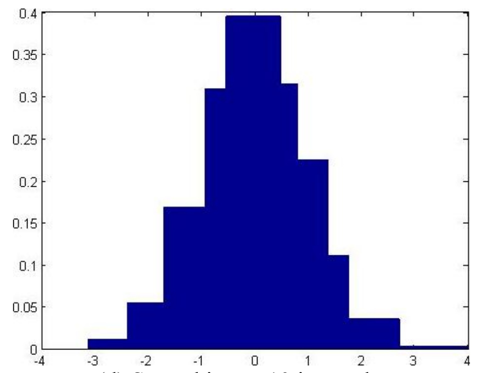
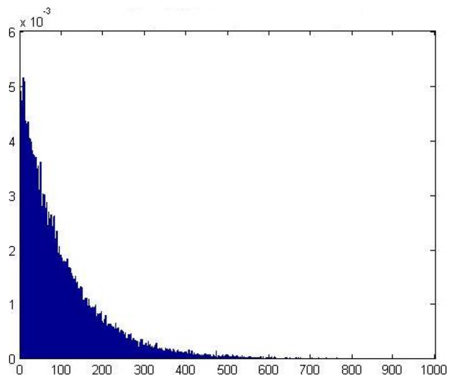
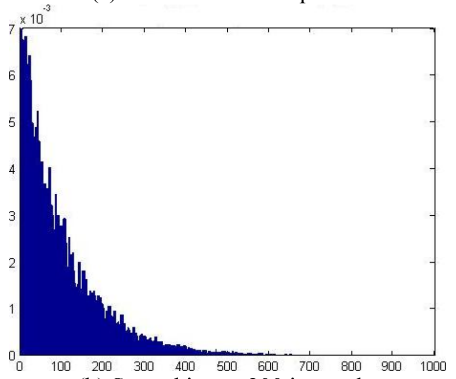
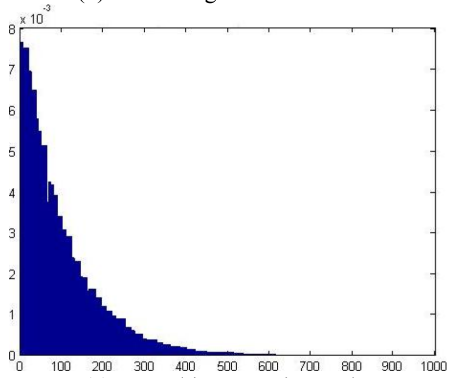

(d) Smoothing to 10 intervals.

Figure 9: Smoothing for the *normal distribution*. We have randomly selected 5000 points based on the normal distribution with the mean 0.0 and standard deviation 1.0 (a), and generated three smoothed PU-distributions with different number of intervals (b–d).


(a) Initial set of 5000 points.


(b) Smoothing to 200 intervals.


(c) Smoothing to 50 intervals.

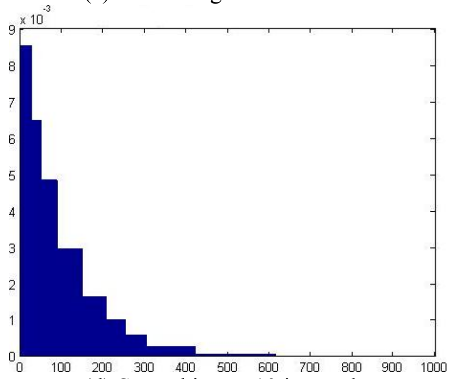

(d) Smoothing to 10 intervals.

Figure 10: Smoothing for the *geometric distribution*. We have randomly selected 5000 points based on the geometric distribution with $p = 0.02$ (a), and generated smoothed PU-distributions (b–d).

We have measured the speed of the smoothing procedure, implemented in C++, on a 3.4GHz Pentium computer. We give the results in Figure 8, which confirm that the "practical" running time is $O(m)$, more specifically, between $2 \cdot 10^{-3} \cdot m$ and $3 \cdot 10^{-3} \cdot m$ milliseconds. For example, the smoothing of 500,000 points takes 1.4 seconds. We have not observed any speed difference among different distributions.

VIII. Concluding remarks

We have described a greedy procedure for constructing an approximate probability density function based on a given point set or a fine-grained histogram. It allows control over (1) the smoothing level, specified by the target number of intervals in the compressed histogram, and (2) the distance function that defines similarity between the original and the compressed histogram. A related future challenge is to propose a more robust representation for approximating probability density functions, which would still support efficient smoothing. Another challenge is to address the problem of constructing globally optimal smoothed distributions.

Acknowledgments

We are grateful to Bin Fu for his help with implementing the described technique. We also thank Helen Mukomel for her detailed comments, which helped to focus the presentation.

References

[Bardak et al., 2006a] Ulas Bardak, Eugene Fink, and Jaime G. Carbonell. Scheduling with uncertain resources: Representation and utility function. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 1486–1492, 2006.

[Bardak et al., 2006b] Ulas Bardak, Eugene Fink, Chris R. Martens, and Jaime G. Carbonell. Scheduling with uncertain resources: Elicitation of additional data. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 1493–1498, 2006.

[Boneva et al., 1971] Liliana I. Boneva, David Kendall, and Ivan Stefanov. Spline transformation: Three new diagnostic aids for the statistical data analysis. Journal of the Royal Society, 33, pages 1–17, 1971.

[Fink et al., 2006] Eugene Fink, P. Matthew Jennings, Ulas Bardak, Jean Oh, Stephen F. Smith, and Jaime G. Carbonell. Scheduling with uncertain resources: Search for a near-optimal solution. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 137–144, 2006.

[Fu et al., 2008] Bin Fu, Eugene Fink, and Jaime G. Carbonell. Analysis of uncertain data: Tools for representation and processing. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pages 3256–3260, 2008.

[Gershman et al., 2009a] Anatole Gershman, Eugene Fink, Bin Fu, and Jaime G. Carbonell. Analysis of uncertain data: Evaluation of given hypotheses. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2009.

[Gershman et al., 2009b] Anatole Gershman, Eugene Fink, Bin Fu, and Jaime G. Carbonell. Analysis of uncertain data: Selection of probes for information gathering. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2009.

[Kullback, 1987] The Kullback–Leibler distance. The American Statistician, 41, pages 340–341, 1987.

[Kullback and Leibler, 1951] Solomon Kullback and Richard A. Leibler. On information and sufficiency. Annals of Mathematical Statistics, 22, pages 79–86, 1951.

[Lin, 1991] Jianhua Lin. Divergence measures based on the Shannon entropy. IEEE Transactions on Information Theory, 37(1), pages 145–151, 1991.

[Parzen, 1962] Emanuel Parzen. On estimation of a probability density function and mode. Annals of Mathematical Statistics, 33(3), pages 1065–1076, 1962.